

SECURE COMPUTERS and NETWORKS

Analysis, Design, and Implementation

Eric A. Fisch, Ph.D.

KPMG LLP
Information Risk Management
Dallas, Texas

Gregory B. White, Ph.D.

SecureLogix
San Antonio, Texas



CRC PRESS

Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Fisch, Eric A.

Secure computers and networks : analysis, design, and implementation / Eric A. Fisch,
Gregory B. White.

p. cm.

Includes bibliographical references and index.

ISBN 0-8493-1868-8 (alk. paper)

1. Computer security. 2. Computer networks—Security measures. I. White, Gregory B.

II. Title.

QA76.9.A25 F5334 1999

005.8—dc21

99-052126

CIP

This book contains information obtained from authentic and highly regarded sources. Reprinted material is quoted with permission, and sources are indicated. A wide variety of references are listed. Reasonable efforts have been made to publish reliable data and information, but the author and the publisher cannot assume responsibility for the validity of all materials or for the consequences of their use.

Neither this book nor any part may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, microfilming, and recording, or by any information storage or retrieval system, without prior permission in writing from the publisher.

The consent of CRC Press LLC does not extend to copying for general distribution, for promotion, for creating new works, or for resale. Specific permission must be obtained in writing from CRC Press LLC for such copying.

Direct all inquiries to CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431.

Trademark Notice: Product or corporate names may be trademarks or registered trademarks, and are used only for identification and explanation, without intent to infringe.

Visit the CRC Press Web site at www.crcpress.com

© 2000 by CRC Press LLC

No claim to original U.S. Government works

International Standard Book Number 0-8493-1868-8

Library of Congress Card Number 99-052126

Printed in the United States of America 3 4 5 6 7 8 9 0

Printed on acid-free paper

To the woman who makes anything possible,
my beautiful bride, Doreen.

E. A. F.

To my wife Charlan, and our kids, Josie, Heather,
and Gregory:

Thanks for your understanding and encouragement.

G. B. W.

CONTENTS

1 FUNDAMENTALS OF COMPUTER SECURITY

- 1.1 Objectives of Computer Security
- 1.2 Issues Involved in Computer Security
- 1.3 Privacy and Ethics
- 1.4 Computer Crime
- 1.5 Projects
- 1.6 References
- 1.7 Extended Bibliography

2 RISK ASSESSMENT AND MITIGATION

- 2.1 Assessment Theory
 - 2.1.1 Information Asset Value (A)
 - 2.1.2 Vulnerability Evaluation (V)
 - 2.1.3 Threat Measurement (T)
- 2.2 Applying the Risk Analysis Equation
- 2.3 Decision Support and Risk Mitigation
- 2.4 Summary
- 2.5 Projects
- 2.6 References
- 2.7 Extended Bibliography

3 DEVELOPING SECURE COMPUTER SYSTEMS

- 3.1 External Security Measures
- 3.2 Structure of a Computer System
- 3.3 Secure Computer System Issues
- 3.4 Summary
- 3.5 Projects
- 3.6 References
- 3.7 Extended Bibliography

4 SECURITY MODELS

- 4.1 Specification and Verification
- 4.2 Security Models
 - 4.2.1 Biba

- 4.2.2 Bell and Lapadula
- 4.2.3 Clark-Wilson
- 4.2.4 Goguen-Meseguer
- 4.3 The Trusted Computer System Evaluation Criteria
- 4.3.1 Discretionary Access Requirements
- 4.3.2 Mandatory Access Requirements
- 4.4 Summary
- 4.5 Projects
- 4.6 References
- 4.7 Extended Bibliography

5 USER AUTHENTICATION

- 5.1 Authentication Objectives
- 5.2 Authentication Methods
- 5.2.1 Informational Keys
- 5.2.2 Physical Keys
- 5.2.3 Biometric Keys
- 5.3 Summary
- 5.4 Projects
- 5.5 References
- 5.6 Extended Bibliography

6 ACCESS AND INFORMATION FLOW CONTROLS

- 6.1 File Passwords
- 6.2 Capabilities Based
- 6.3 Access Control Lists
- 6.4 Protection Bits
- 6.5 Controls for Mandatory Access
- 6.6 Trojan Horses
- 6.7 Summary
- 6.8 Projects
- 6.9 References
- 6.10 Extended Bibliography

7 AUDITING AND INTUSTION DETECTION

- 7.1 Audit Trail Features
- 7.2 Intrusion Detection Systems
- 7.2.1 User Profiling
- 7.2.2 Intruder Profiling

- 7.2.3 Signature Analysis
- 7.2.4 Action Based
- 7.2.5 Ides
- 7.2.6 MIDAS
- 7.2.7 Haystack
- 7.3 Network Intrusion Detection
- 7.3.1 Network Attack Characteristics
- 7.3.2 NSM
- 7.3.3 DIDS
- 7.3.4 NADIR
- 7.3.5 CSM
- 7.4 Monitoring and the Law
- 7.5 Summary
- 7.6 Projects
- 7.7 References
- 7.8 Extended Bibliography

8 DAMAGE CONTROL AND ASSESSMENT

- 8.1 Damage Control
- 8.1.1 Inform the Authorities
- 8.1.2 Backup System Data
- 8.1.3 Remove the Intruder
- 8.1.4 Contain and Monitor the Intruder
- 8.1.5 Lock Stolen User Accounts
- 8.1.6 Require Additional Authentication
- 8.2 Damage Assessment
- 8.2.1 Attack Recovery
- 8.2.2 Damage Prevention
- 8.3 Summary
- 8.4 Projects
- 8.5 References
- 8.6 Extended Bibliography

9 DATABASE SECURITY

- 9.1 Database Management System Primer
- 9.2 Dbms Vulnerabilities and Responses
- 9.2.1 Inference
- 9.2.2 Aggregation
- 9.2.3 Data Integrity
- 9.2.4 Trojan Horses

- 9.3 Summary
- 9.4 Projects
- 9.5 References
- 9.6 Extended Bibliography

10 NETWORK SECURITY

- 10.1 Network Fundamentals
- 10.2 Network Security Issues
 - 10.2.1 Basic Network Security Objectives and Threats
 - 10.2.2 Security Services
- 10.3 The Trusted Network Interpretation
 - 10.3.1 TNI Security Service
 - 10.3.2 AIS Interconnection Issues
- 10.4 Distributed Systems Security
- 10.5 Modem Access to the Network
- 10.6 Summary
- 10.7 Projects
- 10.8 References
- 10.9 Extended Bibliography

11 SECEURE ELECTRONIC COMMERCE

- 11.1 Certificate Authorities
- 11.2 Smart Cards
- 11.3 Interesting Applications of Electronic Commerce
 - 11.3.1 Home Banking
 - 11.3.2 Stock Brokerages
 - 11.3.3 Gambling
 - 11.3.4 The Shopping Mall
 - 11.3.5 Other Applications
- 11.4 Digital Cash
- 11.5 Trusting the Web
- 11.6 Summary
- 11.7 Projects
- 11.8 References
- 11.9 Extended Bibliography

12 WORLD WIDE WEB (WWW) SECURITY

- 12.1 Browser Security
 - 12.1.1 User Certification
 - 12.1.2 Cookies

- 12.1.3 User Privacy
- 12.2 Scripts
- 12.2.1 ActiveX
- 12.2.2 Java and Javascript
- 12.3 Protocols
- 12.4 Summary
- 12.5 Projects
- 12.6 References
- 12.7 Extended Bibliography

13 FIREWALLS

- 13.1 Simple Damage Limiting Approaches
- 13.2 Network Firewalls
- 13.2.1 Packet Filtering Gateways
- 13.2.2 Circuit Level Gateways
- 13.2.3 Application Level Gateways
- 13.3 Firewall Costs and Effectiveness
- 13.4 Sample Security Packages
- 13.5 Summary
- 13.6 Projects
- 13.7 References
- 13.8 Extended Bibliography

14 CRYPTOGRAPHY

- 14.1 Substitution Ciphers
- 14.1.1 Caesar Cipher
- 14.1.2 ROT13
- 14.1.3 Substitution Cipher Variations
- 14.1.4 Vigenere Ciphers
- 14.1.5 One-Time Pads
- 14.2 Transposition Ciphers
- 14.3 Encrypting Digital Communication
- 14.3.1 DES
- 14.3.2 IDEA
- 14.3.3 Key Escrow and Key Recovery
- 14.3.4 Public Key Cryptography

- 14.3.5 [Digital Signatures](#)
- 14.4 [PGP – Pretty Good Privacy](#)
- 14.5 [Public Key Infrastructure](#)
- 14.6 [Steganography](#)
- 14.7 [Summary](#)
- 14.8 [Projects](#)
- 14.9 [References](#)
- 14.10 [Extended Bibliography](#)

15 MALICIOUS CODE

- 15.1 [Viruses](#)
 - 15.1.1 [Infection](#)
 - 15.1.2 [Theory behind Viruses](#)
 - 15.1.3 [Prevention, Detection, and Removal](#)
 - 15.1.4 [Special Viruses](#)
- 15.2 [Worms](#)
 - 15.2.1 [Infection](#)
 - 15.2.2 [Theory of Worms](#)
 - 15.2.3 [Prevention and Removal](#)
- 15.3 [Trojan Horses](#)
 - 15.3.1 [Receiving Trojan Horses](#)
 - 15.3.2 [Theory of Trojan Horses](#)
 - 15.3.3 [Prevention, Detection, and Removal](#)
- 15.4 [Summary](#)
- 15.5 [Projects](#)
- 15.6 [References](#)
- 15.7 [Extended Bibliography](#)

16 SECURITY STANDARDS

- 16.1 [The History of Security Standards](#)
- 16.2 [The Trusted Computer System Evaluation Criteria](#)
- 16.3 [The Information Technology Security Evaluation Criteria](#)
- 16.4 [The Canadian Trusted Computer Product Evaluation Criteria](#)
- 16.5 [The Federal Criteria](#)
- 16.6 [The Common Criteria](#)
- 16.7 [British Standard 7799](#)
- 16.8 [Summary](#)
- 16.9 [Projects](#)
- 16.10 [References](#)

16.11 [Extended Bibliography](#)

17 CASE STUDIES

- 17.1 [The Hanover Hackers](#)
- 17.2 [An Evening with Berferd](#)
- 17.3 [The Internet Worm](#)
- 17.4 [Adventures on the World Wide Web](#)
- 17.5 [Summary](#)
- 17.6 [Projects](#)
- 17.7 [References](#)
- 17.8 [Extended Bibliography](#)

APPENDIX A: INFORMATION WARFARE

- A.1 [Levels of Information Warfare](#)
- A.2 [Weapons of Information Warfare](#)
- A.3 [Perception Management](#)
- A.4 [Summary](#)
- A.5 [Projects](#)
- A.6 [References](#)
- A.7 [Extended Bibliography](#)

APPENDIX B: UNIX SECURITY

- B.1 [History](#)
- B.2 [System Boot](#)
- B.3 [Audit Features](#)
- B.4 [Passwords and Accounts](#)
- B.5 [The UNIX File System](#)
- B.6 [Startup Files](#)
- B.7 [UNIX and Network Security](#)
- B.8 [Sources of Help](#)
- B.9 [References](#)

PREFACE

The world is changing—growing smaller. With the changing business and financial environments, companies are no longer limiting themselves by geographic boundaries. The recent introduction of a new European currency and the recent discussions of stock market mergers have only made global transactions more important to a business' success. This has escalated the need for and dependence upon communications and other advanced technologies, such as the Internet. This, in turn, has increased the need for computer and communication security, and for people that can “speak” it. Additionally, it is important for the non-specialist to understand the basics of security. Terms such as *firewall* and *encryption* are a part of daily communication. For these reasons, we are very excited about publishing this security primer.

The objective of this text is to provide a source suitable not only for a college course or security professionals in the field, but for those with a passing interest in security as well. Computer security, which was once considered overhead to a company's bottom line, is now recognized as an important aspect of business operations. The public perception of a computer intrusion has also changed. What was once seen as just a nuisance perpetuated by high school students and targeted at government agencies, is now being recognized as serious criminal activity. No longer does one have to go to technical or trade journals to read about computer security incidents; their impact can be read about every day in most major newspapers and periodicals.

Even with a greater public understanding of computer security, there is still much misconception about what it entails, how much is needed, and what is appropriate. While the danger of threats such as viruses or hackers is indeed real, the picture is not as bleak as some may perceive. There are a number of basic actions that can be performed to protect computer systems and networks from all but the most talented and persistent of intruders. While some may claim this is not enough—that we need to be able to completely protect our systems—this is neither realistic nor necessary. As we accept a certain amount of latitude in our physical environment, the same should be

true for our “cyber-environment.” For example, one locks their front door to keep burglars out of their home, but a determined thief could still drive a car through the walls to gain access. We accept this level of risk because it is relatively unlikely to occur. The same should be true of our computer systems and networks. The problem we face concerns understanding the issues surrounding security, such as the technology, the risk, and the value of the information we wish to protect. It is these problems that are addressed in this text.

This book is suitable as a general reference on computer security for the security practitioner or as a general text for a course on system security. This book is designed to offer a basic understanding of the issues and elements involved in securing computer systems and networks. A basic understanding of computer operating systems is strongly recommended as background. An understanding of, or experience with the UNIX operating system would also be useful as a number of the examples cited involve this widely used system.

Chapter 1 of the text provides an introduction to the field of computer security and some of the relevant issues. This general introduction continues in Chapter 2 with a discussion of the threats to computer systems, the risks that are posed by these threats, and the use of risk analysis for determining the appropriate protections to employ.

Chapter 3 provides a background to what is required to have a secure computer system and how such a system is designed. Chapters 4 through 7 discuss, in detail, several of these components: Security Models, User Authentication, Access and Information Flow Controls, and Auditing and Intrusion Detection.

Chapter 8 introduces a topic for which relatively little has been written (in terms of automated approaches to the problem and its solution) but which is concerned with one of the most important aspects of an organization’s security policy, Damage Control and Assessment.

Chapter 9 discusses the issues surrounding Database Security. With more and more information becoming available via computers and the Internet, this has become an ever-increasing security and privacy issue.

Chapter 10 addresses the topic of Network Security and how it differs from single host security. Securing networks is not just a matter of extending the practices used for one machine to many but also entails additional elements which are not a concern at the single host level.

Chapter 11 discusses the growing area of Secure Electronic Commerce. Society is growing toward an on-line marketplace, this chapter addresses some of the relevant issues around doing business on the Internet

Chapter 12 follows the discussion of secure electronic commerce with an investigation of World Wide Web (WWW) security. This includes a section on browser security, browser technologies, and server security. Key issues in this chapter are the use of scripts and controls that make surfing the Internet more exciting, and dangerous.

Chapter 13 discusses one method in wide use to secure computer systems and networks which are connected to large, wide area networks and internetworks such as the Internet. This method involves the construction of a Firewall that will filter the traffic allowed to access a network or individual host.

Chapter 14 provides a background to cryptography. It begins with a discussion of general purpose cryptographic techniques and then discusses the application of cryptography to digital communications.

Various types of Malicious Software are discussed at several points in the text. Chapter 15, however, goes into more depth in this area of concern. Viruses, Worms, and Trojan Horses are examined as well as methods to address each of these threats.

Chapter 16 provides a brief explanation of the various governmental security standards found in use today. Their history, application, and probable future are also addressed.

Chapter 17 presents studies of several incidents that have occurred which illustrate the problems and potential for harm when security fails. They also serve illustrate the difficulty in protecting against concerted attacks and in discovering the culprits behind such attacks.

Finally, we close the text with two appendices on some of the more “hot topics”—Information Warfare and UNIX Security. *Information Warfare* refers to what many believe to be the newest form of combat and which some believe will be the warfare of choice for terrorists in the future. Because of the variances between many of the different UNIX flavors, this appendix only addresses some of the more fundamental issues associated with UNIX systems.

In preparing this text, we have attempted to provide a broad coverage of the topics associated with computer and network security. We have not gone into to significant detail on any single subject. Indeed, each chapter could be extended to become a textbook in itself. Instead, we have included the detail necessary to provide the readers with an understanding of the problems associated with the many aspects of security. We include at the end of each chapter not only a list of the references cited in the chapter but an extended bibliography as well. We chose not to place all citations in a section at the end of the book as we wanted to provide the readers the ability to easily find useful references for a specific topic they might find of interest.

We close this Preface with an acknowledgment of those who have helped make this book possible. Many individuals have helped contribute to this text and we would be remiss if we were to not take the time to recognize some of them. We offer a special thanks to Dr. Udo W. Pooch at Texas A&M University for his support and guidance on this exciting endeavor. Additionally, we wish to thank our publisher, Dawn Mesa, for her unyielding patience and assistance. We need to also our employers, KPMG LLP and the SecureLogix Corporation for their implicit support and stimulating environment in which to write this book. Our final acknowledgments go to the many researchers, practitioners, and colleagues who have contributed to the

development of this fast growing field. As always we thank all contributors, named or otherwise, while acknowledging our responsibility for the accuracy of that which has been included in this text.

Eric A. Fisch
Gregory B. White

1

FUNDAMENTALS OF COMPUTER SECURITY

Computers have become commonplace in today's society. They are used in banking for everyday actions such as Electronic Funds Transfer (EFT) and Automated Teller Machine (ATM) transactions. They are used to store a wide range of information about us such as medical, credit and financial data. They are used to help fly the commercial aircraft we travel in and to operate the cars we drive. They store trade secrets for corporations and military and diplomatic secrets for governments. They are used to help control our telephone communication networks and to process our paychecks. It can be truly said that everyone's life has somehow been touched by a computer.

With this tremendous amount of interaction comes a comparable level of responsibility for those who control these computers. It is of the utmost importance that the records the computers contain paint an accurate picture of who we are. We have probably all heard stories about individuals who have had their credit rating inadvertently affected by a mistake in one of these systems. Mistakes of this kind can affect our ability to buy a home or car, or can even lead to legal actions against us until the mistake has been corrected. Even more important than the computers that control our financial well-being are those that are used in critical applications that can affect our physical well-being. The modern hospital is filled with advanced medical equipment run by or with the aid of computers. Should one of these machines malfunction, it could result in a loss of life.

The unintentional error in a program or entry in a database are not the only problems that we must worry about we must also be concerned with the intentional misuse of these computer systems. Dishonest employees may try to modify account information in order to funnel funds or products to themselves. Companies may attempt to access marketing plans and trade secrets of rivals in order to gain a sales advantage. Individuals who feel they have been treated unfairly or have been offended in some way may attempt to seek revenge by attacking another person's financial or credit

records. It is these and similar problems that Computer and Network Security is concerned with.

1.1 Objectives of Computer Security

When computers were first developed, computer security was simply a matter of providing the physical protection mechanisms to limit access to all but a few authorized individuals. With today's worldwide networks, however, computer security involves much more. Despite its expanded nature and increased importance, computer security today has the same basic objectives as forty years ago. The three fundamental objectives of computer security are:

- Confidentiality
- Integrity
- Availability

Confidentiality requires that the data in a computer system, as well as the data transmitted between computer systems, be revealed only to authorized individuals. This may not only include protection from unauthorized disclosure of the actual data, but the fact that certain data even exists. The fact that an individual has a criminal record, for example, is often just as important as the details of the crime committed.

Integrity stipulates that the data in a computer system, as well as the data transmitted between computer systems, be free from unauthorized modification or deletion. It also includes the unauthorized creation of data. The unauthorized insertion of false credit records, for example, could jeopardize an individual's ability to obtain credit. It is important that records such as these are only created, modified, or deleted by authorized individuals and that this occurs in a prescribed manner.

The objective of availability requires that the authorized users of the computer systems and communications media not be denied access when access is desired. This objective is also associated with the concept of *denial of service* which is manifested by a reduction in system performance. This does not include normal degradation of the system performance during peak operating periods but rather specific acts taken by attackers to influence the ability of authorized users to access the system.

Most research in computer security has been in the area of confidentiality. The historical reason for this is that the majority of funding for computer security has been supplied by the federal government whose chief concern has always been maintaining the secrecy of its classified documents. The problems caused by the destruction or modification of data has always taken a back seat to the one of disclosure. Fortunately for proponents of integrity issues, the safeguards and techniques used to implement

confidentiality are closely related to that of integrity. If a person can't see the data, it will generally be hard for them to destroy or modify it as well.

In addition to the three fundamental objectives already mentioned, several other secondary objectives are frequently listed including, ***authorized use***, ***message authentication***, and ***non-repudiation***. Authorized use simply means that only authorized individuals may use the computer system or its peripherals and then only in a prescribed manner. Message authentication and nonrepudiation are both associated with the widespread use of computer networks. Often when a message is received we want to be sure that the individual who the system claims sent the message did indeed transmit it. This is message authentication. At other times we want to know that an individual did receive a message that was transmitted. This is nonrepudiation. Taken together, all of these objectives serve to provide the needed foundation for computer and network security.

1.2 Issues Involved in Computer Security

The objectives of computer security seem simple enough yet a foolproof implementation still eludes us. The reason for this is that, fundamentally, securing a computer system is a complex task. There are several factors which make securing a computer system or network hard. These include:

- Secure operating systems involve a tremendous amount of software and large software projects have historically proven to be nearly impossible to implement error-free.
- Security is often not included in the originally designed or implemented system but is added later in the project.
- Security costs and often “gets in the way”.
- Very often the problem lies with the people who use the system and not in the technology.

The first issue is a common one in computer science. Anyone who has ever written software knows how hard it is to create a program that is error-free. The larger the program the more this is true. For a ‘normal’ program, the existence of a few bugs can generally be tolerated as the users simply learn to live with the problem or to somehow work around them. In security, however, the existence of a single error can result in a hole through which intruders can gain access to the system. This is clearly not acceptable. For security then, the existence of a single error is often fatal. In addition, an intruder does not have to find all holes that exist in an operating system in

order to break in, only one hole is required. The programmer responsible for the operating system, however, needs to worry about all holes in order to plug them.

The second issue is a financial one. Most projects in the government and industry operate under very tight budgetary constraints. When the purchase or development of a computer system is contemplated, the chief concern will be whether the system will be able to accomplish the task it was intended to perform. Secondary concerns generally are centered around issues such as how much will the system cost and how fast will it accomplish the required task. Seldom is security considered. In fact, security is often not considered until later when the occurrence of a security incident forces the issue. Attempting to retrofit security is an expensive process, in terms of both money and labor.

Another issue in implementing security is that it is often viewed as ‘getting in the way’ of the user. For example, many computer operating systems provide the capability to record the actions of all users on the system. The resulting *audit trail* may occupy a tremendous amount of disk space and recording the actions, especially if any analysis of the data is performed, takes valuable computer time away from other processes. This security feature is thus often viewed as an expensive overhead that can be done without. Another example of a security feature that is often viewed by users as bothersome is passwords. Passwords are used to control access to the computer system and its data. They are analogous in many respects to a combination for a safe and just like the combination, they are frequently hard to remember. Often users are allowed to select their own password. This leads to an interesting dilemma. If a user picks a password which is easy to remember, it is probably also easy for an intruder to guess. This defeats the purpose of the password in the first place. If, on the other hand, a totally random sequence of characters is chosen for the password it is hard to guess but also hard for the authorized user to remember. If we make it easy for the authorized users we make it easier for the intruders. If we make it hard for the intruders we also make it hard for the authorized users. Thus security, in terms of passwords, is often viewed as either worthless or cumbersome.

One final issue that must be considered in any discussion on computer security is that often the problem is not technology, but people. The majority of computer crimes committed by ‘insiders’ (i.e., authorized users) do not involve any violation of the system’s security rules. Instead they involve an abuse of the individual’s authority which has been granted in order for them to perform their assigned job. This can be illustrated by examining what occurred in one office of an agency of the Federal Government. An employee discovered that the computer system she worked with would allow an individual who had been receiving benefits to be “resurrected” should the individual inadvertently be listed as deceased (through some clerical error). The “resurrected” individual could then be issued a special check to retroactively provide for the benefits that should have been received while the individual was listed as deceased. The employee decided to take advantage of the system and collected a series of names of

people who had been dead for at least five years. She then used the resurrection feature to generate a check for each, which she had sent to her own post office box. After the check was issued, she changed the individual's records back to show them as deceased [1.1]. This was not a case of an unauthorized intruder gaining access to a system to perpetrate a crime but rather an authorized individual abusing her authorized permissions. To prevent this sort of crime involves a different approach to computer security than does protecting a computer system or network from an unauthorized individual attempting to gain access.

1.3 Privacy and Ethics

An issue related to computer security provides an interesting paradox involving the joint concerns of privacy and ethics. One of the reasons we are so concerned with the security of computer systems is to maintain the privacy of the individuals whose records the computers maintain. This is a reasonable desire which few, if any, would argue with. If we lived in a society where everyone acted in an ethical manner we would not have a problem. Unfortunately, we live in a society where the actions of a few require certain precautions. This is really no different than the fact that we must lock our homes because a small percentage in our society would take advantage of a more trusting environment. The paradox in computer security occurs when we try to enforce what individual ethics have failed to do. To illustrate this point, consider a common technique used by administrators. A method suggested to ensure the confidentiality and integrity of individual records is to monitor the actions of those who have access to the system. The extent of this monitoring for security purposes has sometimes extended so far as to include the reading of an individual's electronic mail to ensure that no unauthorized activity is occurring. The action of looking through an individual's U.S. Postal Service mail is strictly regulated by law and can only be done under very specific circumstances. Many, however, see no problem with the monitoring of an individual's electronic mail. We must be sure that we are not enforcing the privacy of the records maintained on the system at the cost of the privacy of the users. It should be noted that this is a drastic simplification of a very complex issue. Nevertheless, as we consider the various techniques discussed in this text, we should also consider our actions and how, in the name of security, these actions affect the rights and privacy of all individuals involved.

1.4 Computer Crime

One of the reasons computer and network security has received so much attention recently is the increasing number of reports of computer related crime we read about in the press. Incidents such as the attack on Pentagon computer systems in early 1998 by two teenagers from California and their mentor, an 18-year old from Israel who called himself the ‘analyzer’, served to bring computer intrusions to the attention of the general public. The timing of this specific incident added to its notoriety because the attacks, made against defense information infrastructure hosts, came at a time when the U.S. military was preparing for large-scale air and missile strikes against targets in Iraq [1.1]. While computer intrusions (commonly referred to as “hacking”) into government-owned systems by teenagers seem to receive an indordinate amount of attention, there are many other reasons why computer systems are prime targets in this electronic age. Government systems may be targeted by agents of hostile intelligence services. They may also be attacked by individuals opposed to the government for any number of reasons (e.g., terrorists). Businesses may find themselves attacked by competitors seeking to gain information that would provide them an edge in a tightly contested market. Banks and financial institutions are prime targets and we have seen a rise in the attention organized crime is paying to them. The amount of money transferred electronically everyday is astounding and tapping into that stream to siphon off even a small fraction of the flow would equate to hundreds of millions of dollars.

One of the reasons that computer crime is so attractive is because of the seemingly low risk and the, up to recently, lack of laws governing this area. The Computer Fraud and Abuse Act of 1986 attempted to define penalties for certain acts of computer intrusion and abuse. This act has subsequently gone through numerous revisions in an attempt to clarify various points. It is, for example, a crime, punishable by up to five years in prison, to knowingly transmit a series of commands or a program that causes damage to a “protected computer” (one which is owned by the government or a financial institution or one used in interstate or foreign commerce) [1.4]. It is also a crime, punishable up to ten years in jail, to knowingly access a government computer containing information of importance to “national security” [1.4]. A second such offense, after a conviction for the first, is punishable up to twenty years in jail. Obviously the U.S. government has decided to get serious about laws designed to protect our computer systems.

Creating the laws is not, however, sufficient. The courts must be willing to enforce the sentences outlined in the laws. The problem with this is that it is often hard to justify sending a “hacker” to jail for a long period of time when murderers and rapists are being paroled. What real damage has somebody done, for example, if they simply gained access to a computer system and then set up a bulletin board for other hackers? It is true that the company whose machine was accessed may have run slower

as numerous hackers accessed the system. The company may even have lost valuable files if the hackers weren't careful and deleted them as they wandered through the system. But compared to the damage caused by rapists and murderers, the casual hacker is practically harmless. This at least seems to be the attitude taken by the courts up to this point. Consider the case of a Massachusetts teenager who was convicted of disrupting communications in the air-traffic control tower in Worcester, Massachusetts in March of 1997. The U.S. attorney in the case worked with the defense lawyers to come to an agreement which put the teenager on two year's probation, fined him \$5,000, and sentenced him to 250 hours of community service [1.4]. At first it may seem that the sentence was light considering the damage that might have occurred had the control tower not been able to use backup systems, but the teenager's actions were not intended to cause this disruption, they were simply the actions of an individual on a "digital joyride." The court felt leniency was appropriate even though the law allowed for a much stiffer penalty and past court decisions have clearly stated that an individual did not have to intentionally damage a system, just proving that the system was damaged was sufficient for conviction [1.3]. The last chapter has not been written on computer crime or the laws concerning them. As more individuals are convicted and as additional case law is created we will have a much better feel for what new laws need to be created, what sentences are appropriate for individuals convicted of various computer crimes, and what it takes to deter individuals from committing the crimes in the first place.

1.5 Projects

- 1.1 Try to create a list of jobs in today's society that do not somehow involve the use of a computer. Don't forget modern telephone switching is controlled by computers or the more mundane but commonplace applications such as word processors which have all but eliminated the use of typewriters in today's business world.
- 1.2 What are the privacy implications of local merchants maintaining a database of customers and their purchases? What if the business is a grocery store which sells alcoholic beverages or a bookstore which sells pornographic magazines? What if the business rents video tapes? Does it matter if this type of information about an individual becomes public knowledge?
- 1.3 Many of the individuals who break into computer systems never cause any damage but claim to do it only because of the intellectual challenge

involved. In addition, a number claim that they actually are helping computer and network security professionals by revealing holes or weaknesses in security packages. Comment on these assertions.

- 1.4 Computer networks have been used for a variety of illegal activities including the transmission of child pornography. Individuals who send and receive these files often encrypt (disguise) them so it isn't obvious what they contain. Some law enforcement agencies want to restrict the use of encryption schemes so that they can uncover these (and other) criminal activities and to be able to obtain the evidence necessary to convict the individuals involved. Is this a valid request by our law enforcement agencies or is this an infringement on our right to privacy? How is it helpful to consider the rules that govern U.S. Postal Service mail when considering this problem? What about laws governing wiretaps and electronic surveillance?

1.6 References

- 1.1 Martin, Larry, "Unethical 'Computer' Behavior: Who is Responsible?", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 531-541.
- 1.2 Gertz, Bill, "Teen hackers create scare at Pentagon of electronic war", *Washington Times*, April 21, 1998, pg. 3.
- 1.3 United States Code, Annotated, Title 18, Chapter 1030.
- 1.4 Hamblen, Matt, "Security pros: Hacking penalties too lenient", *Computerworld*, March 23, 1998, pg. 2.

1.7 Extended Bibliography

- 1.5 Campbell, Marlene, "Security and Privacy: Issues of Professional Ethics", *Proceedings of the 10th National Computer Security Conference*, Gaithersburg, Maryland, September 1987, pp. 326-333.

- 1.6 DeMaio, Harry, "Information Ethics, A Practical Approach", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 630-633.
- 1.7 Gasser, Morrie, Building a Secure Computer System, Van Nostrand Reinhold, New York, New York, 1988.
- 1.8 Icové, David; Seger, Karl; and VonStorch, William, Computer Crime: A Crimefighter's Handbook, O'Reilly & Associates, Inc., Sebastopol, California, 1995.
- 1.9 Perry, Tekla and Wallich, Paul, "Can Computer Crime Be Stopped?", *IEEE Spectrum*, Vol. 21, No. 5, May 1984, pp. 34-45.
- 1.10 Schou, Corey D., and Kilpatrick, John A., "Information Security: Can Ethics Make a Difference?", *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 305-312.
- 1.11 Shankar, K.S., "The Total Computer Security Problem", *Computer*, Vol. 10, No. 6, June 1977, pp. 50-73.
- 1.12 Turn, R. and Ware, W.H., "Privacy and Security Issues in Information Systems", *IEEE Transactions on Computers*, Vol. C-25, No. 12, December 1976, pp. 1353-1361.

2

RISK ASSESSMENT AND MITIGATION

Risk assessment and mitigation is the process of finding, evaluating, and correcting the potential damage associated with a security breach. The goals of performing a risk assessment are two-fold; to determine the strength of a computer system or network and to make an educated decision as to how the security can, and should, be improved. The benefits of performing a risk assessment are not just the end results of improving security, but the detailed knowledge regarding the system and its flaws [2.1, 2.2]. [Table 2.1](#), below, provides a list of the key information gained by performing a risk assessment. This information is either collected during the assessment and mitigation process or results from the analysis and evaluation involved in a risk assessment.

Table 2.1. Information gained by performing a risk assessment.

- Determination of an organization's sensitive and critical assets
- Identification of threats facing an organization
- Identification of specific system vulnerabilities
- Identification of potential losses
- Identification of effective countermeasures
- Implementation of a cost effective security system

2.1 Assessment Theory

The goals of a risk assessment are to identify the areas of a computer or network that are most susceptible to compromise and to determine the most appropriate protection for those areas. This is accomplished by analyzing three risk attributes: Asset Value, Threat, and Vulnerability. Asset value is the relative importance of information on a computer or network. Information that is less important to the daily business operations of an organization, such as an employee's personal e-mail, should have a low asset value. The organization's current inventory list or payroll data, however, should have a higher asset value. Threat is a means of quantifying an organization's concern regarding the types of attacks that may occur. It allows an organization to focus their security efforts and solutions towards different types of attack sources, such as a disgruntled employee or competitor. The third attribute, vulnerability, is an actual measure of the current technology. It is typically derived from a hands-on review of computer security, network security, and general operating practices. Without any one of these three values, there is no risk. For example, an old clunker of a car equipped with all of the possible leaks and rust spots, parked in the seedy side of town, with the keys in the ignition and an open window has no risk—there is a vulnerability (the keys in the ignition and an open window), and a threat (its in a dangerous neighborhood), it has no asset value (it is worthless). [Table 2.2](#) provides a few additional examples to assist in this explanation.

Table 2.2. Example risk evaluations.

Example Scenario	Asset Rating	Vulnerability Rating	Threat Rating	Overall Risk
An uncovered bag of groceries in the deep woods surrounded by wolves and bears.	High	High	High	High
An empty bag of groceries in the deep woods surrounded by wolves and bears.	Low	High	High	Low
A bag of groceries in an air-tight container in the deep woods surrounded by wolves and bears.	High	Low	High	Low
An uncovered bag of groceries in kitchen pantry.	High	High	Low	Low

The three attributes that comprise risk can be given numeric ratings and combined to produce a relative measure of risk for a specific vulnerability on a specific system (or network) within a specific organization. Equation 2.1 is the foundational measure used to combine these attributes. This equation is one of many that are used throughout the highly mathematical risk assessment process.

$$\textit{Relative Risk} = A \times V \times T \quad (2.1)$$

Equation 2.1 is the foundation of the risk assessment. It is used to compute a **relative risk value** for every combination of asset, vulnerability, and threat. The risk is **relative** because the equation is based upon values that are subjective (relative) rankings and the actual value means little. It is the ranking of these values with respect to each other that drive the risk assessment. For this process to be fully understood, each of the attributed must be discussed in detail.

2.1.1 Information Asset Value (A)

The risk assessment process is focused on protecting an organization's information assets. As such, the information assets included in the assessment process must be identified. These often include the payroll data, the e-mail, client files, internally developed software systems, access control lists, inventory lists, and internal maintenance records. Because of the importance and confidentiality of each asset, they are typically found on separate computer systems within a single network. It is not uncommon to see a payroll server, a mail server, a few production systems, a few development systems, an inventory tracking server, and an internal maintenance server all in a single small organization. The risk assessment process therefore evolves into an examination of each system. This fact, however, becomes of greater importance when evaluating the vulnerability attribute as opposed to assigning asset values.

The information asset value (A) is a number that describes the relative value of a given resource. This value takes into account the cost and time associated with maintaining and administering the resource. Because it is frequently difficult to determine an accurate dollar value of the information and time to maintain or restore corrupt data, a relative rating is used. This is an acceptable alternative because it is not necessary to have an exact appraisal to be able to determine which assets are of greater sensitivity and criticality to an organization.

The asset value is based upon three generally accepted measures of an asset: **confidentiality**, **availability**, and **integrity** [2.4]. **Confidentiality** is the measure of privacy required by an asset. The higher the rating, the more important

confidentiality is to the asset. *Availability* is the measure of access requirements. The higher the rating, the more important it is to have continuous access to the asset. *Integrity* is the measure of asset accuracy and trust. If the data requires a high level of trust and accuracy, it will have a high integrity rating. Some assessment methodologies include additional measures such as *reliability* and *usability*, however, these attributes are often unnecessary because they are combinations of the other three attributes and provide little additional value. The sum of the three attributes, as shown in Equation 2.2, is the asset rating *A* used in Equation 2.1. It also identifies the systems that are most sensitive and critical to the organization. For this number to be meaningful, however, a few basic rules must be followed. First, the scale used for each rating must be the same. It does not matter if the numbers range from 1 to 5 or 1 to 10. Second, this same scale should be used for the threat ratings and vulnerability ratings used in Equation 2.1 as to prevent one of the three ratings from having greater impact than the others. This may require each of the ratings to be scaled before they are combined.

$$\text{Asset Value} = \text{Confidentiality} + \text{Availability} + \text{Integrity} \quad (2.2)$$

2.1.2 Vulnerability Evaluation (V)

The second component of the equation is *V*, the vulnerability ratings. A system is considered to be vulnerable when there is an opportunity for a damage or loss to occur on the system [2.3]. For example, an administrator account without a password is a system vulnerability. For each vulnerability such as this found on the asset's system, an impact value must be assigned. This value indicates the severity of the vulnerability. For example, a vulnerability that grants an unauthorized user system administrator access would receive a high value and a vulnerability that allows a user to read another user's mail would have a lesser value. This process raises questions regarding vulnerability data collection, vulnerability rating, and manipulating potentially unequal amounts of data for each asset.

Vulnerability data collection may occur in many ways. One method is to manually check the system for known vulnerabilities. Because of the inconsistency or potential mistakes that may arise from this approach, a more automated method is preferred. Many of these tests can be combined into a user-written program or script. This is not necessary because there are many public domain tools available on the Internet that perform these tasks. To address the lack of user support and outdated nature of these tools, many companies sell well-maintained tools. Many of the professional tools also perform a necessary task, they rate the vulnerability.

Every vulnerability that is searched for must be given a rating so that it may be used with the risk assessment equations. If the vulnerability examination process is manual, this process will also have to be performed. This can become quite time consuming as each vulnerability will have to be researched in order to determine reasonable ratings. Some automated tools include rating information thus reducing the amount of background work, and potential confusion, necessary to perform the assessment. Any means of reducing the chance of error, especially when large amounts of data are involved, is beneficial.

While each asset, or system, has only one asset value, it can have an unlimited number of vulnerability ratings. Unlike the asset components, these values are not combined into a single value; at least not yet. For example, if there are 20 systems and each has at least 15 vulnerabilities, there are 300 data points to consider. This does not include the data that comes from the threat measurement. This process is best performed and controlled with the use of many automated processes and tools, including a spreadsheet.

2.1.3 Threat Measurement (T)

The last of the three components in the risk assessment equation is threat. In this risk assessment model, threat is defined as the potential for damage—not too different from any other definition. A critical understanding of this definition is that success has no relationship with threat. Threat is strictly the potential for an attack, not a measure of the success of an attack. This is an important distinction because, while it is important to have an idea what number of attacks will be successful, it is just as important to have an understanding of what type of attacks are more likely to occur.

To be able to say which source of an attack is the greatest threat, one must be able to categorize the threat. To begin, there are two major types of threat: environmental and man-made. Environmental threats are things such as hurricanes, tornadoes, and the ever popular volcanic eruption. While these are important to the concept of threat, they are not applicable to this security model. Man-made threats, however, are of significant concern to this model because they represent the potential for criminal electronic operations against an organization. As such, it is the man-made threats that will be the focus of the remainder of this model.

Man-made threats include a wide range of activities from spilling coffee on a diskette to stealing a competitor's intellectual property. These threats can be further divided into hostile activity and non-hostile activity. Hostile activity is described as activity with a criminal or malicious intent, such as corporate espionage. Non-hostile activity is often described as either "curiosity" or "accidental." Examples of non-hostile activity are accidentally unplugging a system, accidentally deleting a file, or the

“ethical” hacker that breaks into computers just to see if it is possible. To provide a more detailed classification of threat, these types of threats must be further divided, not once but twice.

The next distinction is between sophisticated threat and unsophisticated threat—sometimes called structured and unstructured threat. A sophisticated attack comes from a person who is knowledgeable and calculating in their attack. This is typically someone with the skills and resources to execute the attack. For example, a person funded by an organization or government, or a person well versed in “hacking.” The unsophisticated threat is not knowledgeable in the art of computer break-ins and does not know exactly what to do. In some cases, the unsophisticated attacker’s interest will be spurred on because of a news story or other impulsive action. This type of activity is common, but typically unsuccessful. An example of the unsophisticated threat is the person that tries to break into a system with neither the knowledge nor the capability.

The final threat distinction is not based on talent or intention, but location—internal or external. This location is in reference to the network, not the physical office space. An internal attack comes from someone that is connected to their target’s network, often an employee. An external attacker must first gain access to the network, even if he or she is inside the target’s office space. While is a significant hurdle to overcome, it is not insurmountable. In many cases, having to gain access from an external network is not so difficult as to be a disadvantage.

Combining the three different attributes produces eight distinct threats. They are: (1) hostile sophisticated internal threat, (2) hostile sophisticated external threat, (3) hostile unsophisticated internal threat, (4) hostile unsophisticated external threat, (5) non-hostile sophisticated internal threat, (6) non-hostile sophisticated external threat, (7) non-hostile unsophisticated internal threat, and (8) non-hostile unsophisticated external threat. Descriptions and examples of each of these threats are provided in [Table 2.3](#).

As each organization reviews the eight threats, it must determine which threats are more probable and which are less probable to occur (recall, that this does *not* mean successfully occur). For the purposes of calculating threat, each of the eight threats should be rated on the same scale as the assets and vulnerabilities, with the lower value equating to low threat. For example, a financial institution may determine that the threat from hostile sophisticated external attackers is greater than that of the non-hostile unsophisticated external attacker, and rate them 4 and 3, respectively. This process raises two significant issues. First, who determines an organizations threat ratings. Second, how do they differ for each organization.

To rate the potential threat an organization faces with some accuracy, knowledgeable employees must come together. This includes network, security, and system administrators along with high level officers and management. Each

organizational role will typically provide insight not normally considered and thus combine to create a more realistic view of threat. The group must recognize before they begin this process that there is no right or wrong answer, only a model on which to base their decisions.

It is not so astonishing to think that most organizations will have different ratings, however, a pattern may exist within an industry. For example, financial institutions may all rate the internal threats higher than their external counterparts. Military or government organizations may rate the external threats higher than internal, but also rate the sophisticated threats higher as well. When identified, these patterns are helpful in validating the ratings of similar organizations.

Table 2.3. Threat descriptions and examples.

Threat	Description and/or Example
Hostile Sophisticated Internal	Activity of a disgruntled system administrator
Hostile Unsophisticated External	Industrial espionage Activity of a knowledgeable and determined attacker with a vendetta
Hostile Unsophisticated Internal	Activity of a disgruntled non-administrator.
Hostile Unsophisticated External	Attacker with limited capability but strong interest in a specific target (e-mail spam or trial-and-error attacks)
Non-Hostile Sophisticated Internal	Activity of a curious administrator (testing the latest vulnerability reports) Accidental damage by an administrator (deleting system users or files)
Non-Hostile Sophisticated External	Activity of a knowledgeable, determined, and curious attacker (Man vs. Machine)
Non-Hostile Unsophisticated Internal	Activity of a curious user (guessing passwords) Accidental damage by an user (deletion of files)
Non-Hostile Unsophisticated External	Activity of a curious user ("what does this do?") Unintentional activity that may be interpreted as malicious

2.2 Applying the Risk Analysis Equation

The objectives of applying the risk analysis equation are to produce the data necessary to determine how to best proceed with mitigating the risk in the environment. Some key figures used to accomplish this are: asset risk ratings, vulnerability risk ratings, and vulnerability frequencies. This is accomplished by applying Equation 2.1 and building upon the results with additional information, such as cost details and impact details.

Once the assets, vulnerabilities, and threats of an environment are rated, the risk equation can be applied. For every combination of the three values, a measure of risk is created. According to Equation 2.1, this is calculated by multiplying the three values of asset, vulnerability, and threat. This equation can now be re-written as described in Equation 2.3.

$$\text{For each } A, V, \text{ and } T: \text{Relative Risk} = A \times V \times T \quad (2.3)$$

The problem with Equation 2.3 is that for each asset and vulnerability, there are eight threat values that must be considered. It makes little sense to simply sum the eight values and use that for the threat value, since multiplying any constant through the equation has no impact to the final ratings. Before the threat value can be multiplied through, it has to be re-evaluated with respect to the vulnerability in question. As it turns out, not every threat is in a position to exploit every vulnerability. For example, an unsophisticated threat cannot be expected to exploit a highly technical vulnerability. Thus, the value used for threat in Equation 2.1 may vary with each vulnerability. The final threat value for each vulnerability is calculated by summing only the values of the threats that can reasonably be expected to exploit the vulnerability. For example, if a vulnerability requires the attacker to be on the inside and be very knowledgeable, the threat value used with that vulnerability will be the sum of the hostile sophisticated internal threat value and the non-hostile sophisticated internal threat value. This equation modification is shown in Equation 2.4. Manipulating all of these numbers is not so much difficult as it can be confusing. It is also made more complex in that the group performing the assessment will be required to determine which threats are applicable to each vulnerability and ensuring the proper calculations are made throughout the process. Once the proper threat value is determined, the true risk ratings can be calculated.

For each A and V: Relative Risk =

$$A \times V \times \sum T_{APPLICABLE} \quad (2.4)$$

The risk ratings are identified by the name of the specific asset and the name of specific vulnerability. Therefore, each system will have one risk value per vulnerability that affect it, and each vulnerability will have one risk value per asset it impacts. Each of these statistics is important in determining how to address the risks found in the environment.

There are many ways to determine which system's, or asset's, risk should be mitigated first. The simplest method is to calculate the number of vulnerabilities found in each system and prioritize the systems by number of vulnerabilities. This is a good beginning, however, it does not take into account the significance of the system or the danger of the vulnerability and threat—that is were the rating information is helpful. Instead of simply counting the number of vulnerabilities associated with an asset, sum the risk ratings associated with an asset. The end result is a set of numbers that is frequently greater with the number of vulnerabilities, but is also dependent on the environment. For example, an asset with ten low-rated vulnerabilities may have a lower rating (or priority) than an asset with three high rated vulnerabilities, and rightly so. It is not necessarily the asset with the most security risks, rather it is the asset with the most significant risks. This is likewise with the vulnerabilities.

The number of assets a particular vulnerability impacts will assist in determining which vulnerability should be mitigated first. It does not tell you which assets require attention first, but which vulnerabilities within an asset should be addressed first. This frequency value, however, does have the same complication that counting vulnerabilities on a single asset had—it does not accurately identify the vulnerability which most significantly impacts the computing environment. The frequency information is actually best used as a secondary evaluation in selecting a vulnerability. The primary measure of vulnerability impact can either be calculated by taking the highest single rating for each vulnerability or summing the risk rating associated with each single vulnerability, much like the asset calculations

The goal of these, and any other, calculations is to provide a clear understanding of the security environment and how it should be improved. This process should result in a detailed risk mitigation plan with visible and measurable changes. [Table 2.4](#) contains a list of calculations that could be beneficial to an organizations decision support and mitigation process.

Table 2.4. Possible risk calculations.

- Cost/Benefit Analysis
- Vulnerability Descriptions
- Vulnerability Impact Analysis
- Vulnerability Frequencies
- Asset Priority Ratings
- Vulnerability Priority Ratings

2.3 Decision Support and Risk Mitigation

Once every AVT has been calculated, the decision support process and risk mitigation can begin. Decision support is an integral part of this process as an organization's business process will require the security issues to be addressed with respect to the organization's time, resource, and budgetary constraints. Much of this will depend on the calculated risk ratings and frequencies. These values are not the only considerations in selecting security solutions, other constraints invariably exist. Attention must be given to three fundamental constraints: time, fiscal capability, and the business operations of the organizations. If an organization has a limited time frame or budget, some vulnerabilities may go unresolved. Determining which recommendations to delay until money is available is a difficult decision. Clearly some sort of cost/benefit analysis is necessary. This decision process is made more difficult if the most beneficial recommendations hinder the business process. When security prevents people from performing their primary tasks, security will be ignored. While security is important, it is rarely the most important issue to an organization. Decision support, however, is not simply about mitigating system-based vulnerabilities—it is about identifying and correcting the bigger problems associated with the overall operations and management of the computing environment.

Until now, this chapter has been solely dedicated to calculating risk ratings for specific, exploitable vulnerabilities. This is misleading as vulnerabilities at this level are symptoms of larger problems. These problems include insufficient training, inconsistent controls and configurations, a lack of policies, procedures, guidelines, and poor access controls. It is the task of the team performing the risk assessment to determine how each of the vulnerabilities relates to the larger security issues. It is the recommendation and decision of which larger issues to address, and how to address them that are the true benefit of this entire risk mitigation process.

2.4 Summary

Risk assessment and mitigation begins the process of identifying and correcting the security vulnerabilities in an organization's computing environment. It is based on relative ratings assigned to an organization's assets, perceived threats, and identified vulnerabilities. This highly mathematical process produces statistics that assist in determining the order in which assets should be examined and vulnerabilities should be mitigated. These results, however, are not the final answer to the risk assessment and mitigation process; other factors such as time, budget, and business operations must be considered. Furthermore, these results, while extremely valuable, only identify the symptoms of the real security issues in the environment. These issues are typically related to insufficient training, a lack of configuration controls, incomplete policies, and poor access controls.

2.5 Projects

- 2.1 Identify the tools necessary to obtain the vulnerability data from an environment. Rate each of the vulnerabilities that could be found. How many systems need to be examined to obtain an accurate reading of the environment? Justify your answers.
- 2.2 Perform a risk assessment of your environment. What recommendations would you make? Why?

2.6 References

- 2.1 Badenhurst, K. and Eloff, J., "Computer Security Methodology: Risk Analysis and Project Definition", *Computers & Security*, Elsevier Press, New York, **New York** Vol. 9, 1990.
- 2.2 Banks, S., "Security Policy", *Computers and Security*, Elsevier Press, New York, New York, Vol. 9, 1990.
- 2.3 Bennett, S.P., and Kailay, M.P., "An Application of Quantitative Risk Analysis to Computer Security for the Commercial Sector", *The 8th Annual Computer Security Applications Conference*, IEEE Computer Society Press, New York, New York, 1992, pp. 64-73.

- 2.4 Ozier, W., "Risk Analysis and Assessment", Handbook of Information Security Management 1999, CRC Press, Boca Raton, Florida, 1999, pp.425-464.

2.7 Extended Bibliography

- 2.5 Bilbao, Alfonso, "TUAR: A Model of Risk Analysis in the Security Field", *The Carnahan Conference on Security Technology*, IEEE Press, New York, New York, 1992, pp. 65-71.
- 2.6 Drake, David L. and Morse, Katherine L., "The Security – Specific Eight Stage Risk Assessment Methodology", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 441-450.
- 2.7 Garrabrants, W. M.; Ellis III, A. W.; Hoffman, L. J. and Kamel, M., "CERTS: A Comparative Evaluation method for Risk Management Methodologies and Tools", *The 6th Annual Computer Security Applications Conference*, IEEE Computer Society Press, New York, New York, 1990, pp. 251-257.
- 2.8 Lavine, Charles H.; Lindell, Anne M. and Guarro, Sergio B., "The Aerospace Risk Evaluation System (ARiES) Implementation of a Qualitative Risk Analysis Methodology for Critical Systems", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 431-440.
- 2.9 Orlandi, Eugenio, "The Cost of Security", *The Carnahan Conference on Security Technology*, IEEE Press, New York, New York, 1991, pp. 192-196.

3

DEVELOPING SECURE COMPUTER SYSTEMS

Having come to the realization that a computer system requires security measures to protect the data it stores and processes, the next logical step is to determine how to implement these security measures. Early computer systems were made secure through external means such as maintaining the computer system in a locked room. As computer systems added remote users and eventually became connected to other computers and networks of other computers, external security measures were no longer sufficient to maintain security. Internal measures implemented within the hardware and software of the computer system were also needed, though external security measures could not be ignored entirely. Both internal and external measures need to be considered together in order to establish a security policy for a computer system or network.

3.1 External Security Measures

Typical External Security measures include those portions of the total security package for the computer system that do not include the hardware or software of the actual system itself. These measures include:

- Physical Security
- Personnel Security
- Administrative Security

Physical Security measures consist of those techniques used to secure any high-value item including locks, guards, remote surveillance cameras, and alarm systems. These measures are used to not only protect the information that is stored on the computer but to prevent the theft of the computer system or its peripherals.

Obviously, the more important the data or the more expensive the equipment, the more physical controls we will want to employ. If denial of service is of concern, you may also need to consider how an individual bent on denying you the use of your computer system could attack the source of your power or attempt to start a fire at your site in order to activate a fire suppression system. Either attack would bring processing to a halt. While measures to protect against these threats are no different than measures for other high-value pieces of equipment, it is something that we must at least be cognizant of from a security standpoint.

As has been shown on a number of occasions, a system is only as secure as the people who use and operate it. Personnel security is concerned with the procedures used to determine the amount of trust the organization can place in any particular individual. In the military this roughly equates to the security clearance (e.g., Secret, Top Secret etc.) the individual is granted. In a business it can equate to the number of functions an employee is allowed to run (e.g., payroll, shipping,.). In either case, it involves the level of access that the organization has given the individual based on the individual's background, previous experience, and current job requirements. The large percentage of computer crime cases involving insiders illustrates how important it is to pay attention to this aspect of security. There have been numerous cases in which the crime occurred as a result of an employee violating the trust placed in them in order to take advantage of the system for personal gain.

Administrative Security describes the methods to be used to implement the chosen security policies. These statements detail such things as how printouts are to be disposed of, how magnetic media will be stored, erased, and destroyed, what procedures to follow when an employee is fired, and the procedures used for temporary employees or when visitors are in the area. Intruders have often taken advantage of the lack of administrative security (or the poor enforcement of it) to gain access to computer systems and networks. A common trick used by intruders is to call the organization who owns the targeted system posing as an employee of the vendor for the computer system. The intruder will, for example, weave a story involving maintenance problems with the computer system or network, and describe the dire consequences if a certain patch is not installed. To install the patch, however, the intruder will need to be given access for a short while to the system and will request that the individual set up a temporary account granting access. It is amazing how often this simple ploy works. Another common trick is for the intruder to call up posing as somebody else who does have permission to access the system and claiming to have forgotten their password. Often there are policies governing these types of events which are simply ignored or forgotten.

There are many other examples of the types of policies and procedures that should be described by an organization's administrative security mechanism. Consider, for example, the security procedures to be followed when an employee is fired. If an individual is given a two-week termination notice, that individual should immediately

have access to the company's computer system revoked. A considerable amount of damage can be done by a disgruntled employee during a two week period. An example of another policy might entail the use of multi-user accounts. Often several employees will be working on the same project and the temptation is to assign a single account to be used by all members of the project. This cuts down on the overhead involved in granting them access to the system and often simplifies their handling of jointly accessible files. The problem with this is that it makes it hard to provide a level of accountability since there would be no way of telling which individual made a change or deleted a file. In addition, should an employee switch jobs, a new password will have to be issued which must then be passed along to all those who still remain with the project. One final example of an administrative policy is the procedures in place to track the installation of updates or patches to operating system or application software. Often a vendor feature will exist in software that may violate certain local policies. An employee responsible for the operation of the computer system may remove or disable this feature but may not make any note of this. Later, when an update or new version of the software is installed, the same feature will need to be disabled. If no record of the previous actions taken to disable or remove the feature exist, the feature will remain in violation of local security policies. This is especially true if the original employee who initiated the action has transferred to a different job. These examples provide just a glimpse of the large number of factors outside of the actual computer system itself that must be considered.

3.2 Structure of a Computer System

Before we examine the components of a secure computer system it is useful to review the structure of a standard computer system. The parts of a computer system can be divided into two basic types of components, hardware and software (for purposes of discussing security issues, the combination of hardware and software resulting in firmware can be thought of as falling under both categories and will therefore, need to be considered in the discussions for both). The hardware can be roughly divided into Input/Output (I/O) Devices, Secondary Storage, Main Memory, and the Central Processing Unit (CPU) as shown in [Figure 3.1](#). For our purposes, software will be divided into two categories, the Operating System (OS) or just system software, and Application software.

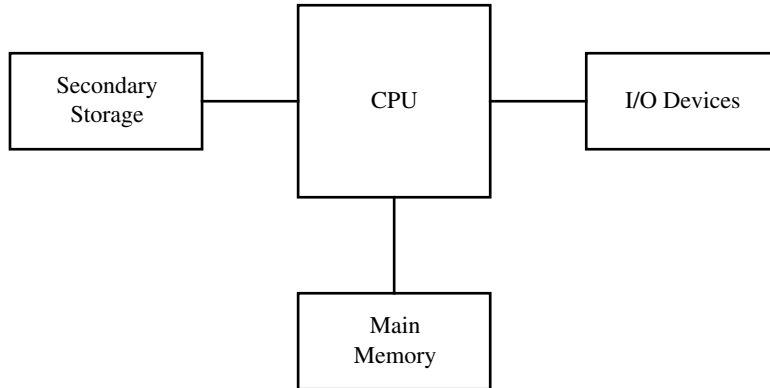


Figure 3.1. Hardware components of a computer system.

A single-processor computer executes one program at any given moment. It must fetch an instruction to be executed from memory, decode it, execute it, store any results, and then repeat this cycle. The operating system is a set of special programs that manages the resources and controls the operation of the computer. In many respects, the operating system serves as a buffer between the application programs and the hardware. The relationship between each of these elements is often represented by a series of concentric rings as depicted in [Figure 3.2](#). Application software consists of those programs designed to perform a task using the resources of the computer system. Examples of application programs include Data Base Management Systems, Word Processors, Spreadsheets, Project Management software, and Communications packages. It will also include any programs written, compiled, and executed by the users themselves.

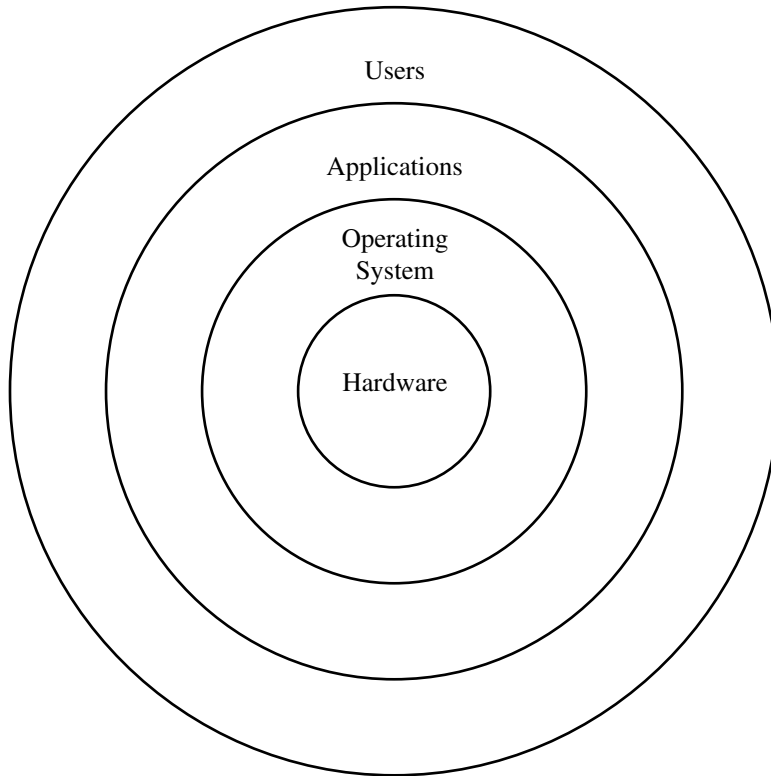


Figure 3.2. Layers in a computer system.

Since much of the time an application program¹ spends may be waiting for interaction with the user or some I/O device, it is useful to have several processes running concurrently. While a single processor can only execute one process at any instant, it is possible to make it appear as if several users have sole control of the machine by dividing the CPU time into segments or slices and providing each user (or process) with a share of the slices. This is known as timesharing. A process which has used up its allotted time will be swapped out by the operating system for another process which is waiting to execute. Since the process being swapped out has not completed executing, the current state of the process must be saved so that it can restart at the point it was interrupted. This saving of the process state is referred to as a

¹ For purposes of this discussion, a program which is currently being executed will be referred to as a process. A single program may be executed multiple times and thus at any given moment have many instances of itself running on the computer. Each one of these will be a separate process competing for resources like every other process.

context switch. After the current state is saved, the next process is loaded, having been restored to the spot at which it had previously been interrupted, and then continues to execute until its time slice has also expired and another process is swapped in. The time any single process gets before being swapped out is generally a small fraction of a second.

Another characteristic of most programs is they tend to execute instructions in blocks of code as opposed to individual instructions located great distances from each other. This means the entire program generally doesn't need to be kept in main memory in order for it to run. Instead, blocks of code can be brought in as needed. The implication is that we can actually have several processes in main memory at the same time with the operating system tasked with the responsibility of keeping track of each process and its various parts. This greatly speeds the process of swapping between programs since they do not need to actually be swapped out to secondary storage. This also, however, makes the operating system's task harder as it maintains control of the various parts of many different programs. One of the concerns of the operating system is to ensure that processes not interfere with other processes by accessing memory outside of the space allocated to them. As it turns out, this is not only important for the smooth operation of the various processes in memory, but also has security implications and is one of the issues that needs to be addressed when designing a secure computer system. It would not be a good idea, for example, to allow one process to access (either for reading or writing) the memory currently owned by another process. In fact, even if a process has been swapped out and a new process brought into memory, it is not generally a good security practice to allow the new process to read the storage locations recently vacated by the other process until the memory has been cleared. Otherwise sensitive data may remain and could be easily read by the next process allowed to access that location in memory.

3.3 Secure Computer System Issues

The hardware structure of a secure computer system is conceptually no different than the one depicted in [Figure 3.1](#). There are, however, a few additional concerns which must be addressed either by the hardware or software. One of these concerns is what is known as *object reuse*. The term object reuse is used to describe the actions that occur whenever one subject gives up an object so it can be reused by another subject. For example, when an individual deletes a file the memory that had been used to store that file can now be used by somebody else to store another file. The security concern revolves around the method the computer system uses to release the memory so that it may be used by another. A good example of this is a PC using the MS-DOS operating system. If an individual uses the *del* command to delete a file, the file isn't

actually destroyed, instead the pointer to it in the directory table is altered so that the file doesn't appear to be present. This allows another user to come along and use the space that was allocated to the original file. If someone were to actually examine the individual bits on the disk, however, they could still access the information that had been stored there before. This is why the ***undelete*** command is able to function. It simply goes to the directory table and changes the entries so that the files once again appear to exist. The security implications of this should be obvious. If a disk which was classified Top Secret has all of its files deleted and then is given to a user who is only cleared for access to Secret information, the user, by simply reading the sectors on the disk, could obtain access to the Top Secret information. A secure computer system will need to insure that whenever an object is released, all information that had been contained on it is cleared so that no trace of it remains for another user to access.

Object reuse applies not only to files but to any object released for use by another subject. Main memory, for example, must also be cleared before it is assigned to another subject and, as was previously mentioned, part of the operating system's task is to insure that the various processes currently executing don't interfere with each other. Let's further examine what would happen if the operating system allowed a process to access portions of memory not currently assigned to it and to write to them. Suppose the section of code the process overwrites contains the password file, for example. This type of problem has actually occurred in systems, most recently in server software used to interact with the World Wide Web (WWW). The specific vulnerability allowed instructions to be pushed onto the program stack by overflowing an internal buffer [3.2]. An intruder could then, by careful selection of the instructions pushed onto the stack, gain unauthorized access to the WWW server.

One way to interject some security control over a computer system is to create what is known as ***security*** or ***execution domains***. Similar to the concentric rings depicted in [Figure 3.2](#), security domains provide a hierarchical structure to the application of security on the computer system. At the center still sits the hardware. Immediately surrounding the hardware is the operating system. The application programs will be split into several different categories. Each category corresponds to another ring in the diagram and allows access to the functions provided by the next inner ring only. An example of this is a database management system which controls and limits access to data from programs that run at higher levels in the hierarchy. Some users might be allowed to read and manipulate data, others only to read data.

This concept of isolating various functions and restricting their access to specific systems can also be applied to the operating system itself. Various designs of secure computer systems have been created which employ this idea. Known as a ***security kernel***, [3.1, 3.3] this method provides a way to develop a secure computing base. The security kernel concept recognizes that modern operating systems are large, complex programs which can be extremely difficult to debug and even harder to ensure that they contain no security holes. The security kernel addresses this by limiting the

protection mechanisms to a few routines which all other routines must use. Thus, instead of attempting to provide security mechanisms in each portion of the operating system, making each portion responsible for maintaining security over the files it manipulates, the security kernel concept defines just a few special programs which will be responsible for security. All other programs must interact with the kernel in order to gain access to the files it protects.

Key to the concept of the security kernel is the abstract notion of a **reference monitor**. The reference monitor enforces security by forcing all subjects (e.g., processes, users) who wish to access an object (e.g., files, portions of memory) to do so only through the monitor itself. Thus, it **monitors** all **references** to objects by subjects. The reference monitor is depicted in [Figure 3.3](#).

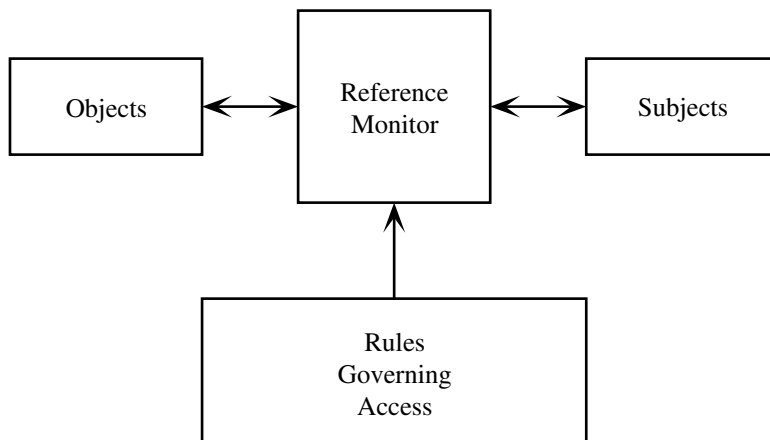


Figure 3.3. The Reference Monitor.

The reference monitor bases its access control decisions on a set of rules which explain what subjects can access which objects. These rules may explicitly specify certain files and objects or, as is more commonly the case, will represent a synopsis of the security policy and model chosen for the system. To be complete, a reference monitor will have to follow three design principles:

- Absolute Control - all access to the objects must be made through the reference monitor.
- Isolation - the kernel must be separated from the other routines and protected from modification or tampering.

- Verifiability - the kernel must be shown to implement some chosen security policy [3.1].

The actual implementation of a security kernel will include both hardware and software concerns. A number of systems have been developed using a security kernel as the basis for their security. The first, and possibly the best, example of this was accomplished by MITRE in 1974 [3.1]. The kernel developed was designed to operate on a Digital Equipment Corporation PDP-11/45 computer system. The kernel consisted of fewer than twenty primitive routines with less than a thousand lines of high-level language code. Despite its small size, the kernel was responsible for the physical resources of the system and enforced protection of the system's objects. This kernel successfully demonstrated both the feasibility of a kernelized design as well as epitomizing the goal of a minimal number of routines being responsible for the security of the system's objects.

Simply implementing a security kernel does not guarantee that the system will be secure. There are many issues that still need to be addressed, such as Security Models, Authentication, Access Control, Auditing, and Cryptography. Security models are chosen to express the security requirements of the system. Authentication involves the determination that an individual is authorized access to the compute system. This involves methods, such as the use of passwords, to verify that the individual is indeed who they claim to be. Authentication is generally thought of as restricting access to the system itself. Restricting access to the objects on the system is the responsibility of access control techniques. Access control techniques determine what objects an individual may access, and what type of access the user has been granted (e.g., read and/or write permissions). Auditing, for security purposes, is a method used to monitor the actions of users in order to determine when intrusive or abnormal system activities occur. Cryptography, which is used to obscure data by encoding it, is a technique commonly used to protect data. All of these other issues are subjects of later chapters in this book.

An interesting problem that we must also be concerned with when designing a secure computer system is the existence of possible *covert channels*. There exist many definitions for a covert channel but for our purposes we will define them as the transmission of information along a channel which is not allowed according to the security policies established for the system. This transmission may either be intentional or unintentional. There are two basic types of covert channels; *covert storage channels* and *covert timing channels*.

A covert storage channel involves the use of a resource that is shared between subjects in two different security domains. The resource (such as a storage location on a disk) will be written to by one subject and read by the other. Normally the sharing of resources between subjects in different security domains is not allowed so in order to communicate the two subjects will need to utilize methods not normally considered for

transmission of information. To illustrate, file names may be readable to everybody on a system while the contents of the file are not. A subject functioning in one security domain (e.g., a user or process operating in a Top Secret environment) might change a file name to send certain information about the file to another subject operating in a different security domain (e.g., a Secret environment). Given sufficient time, a series of name changes could be accomplished which could be used to actually transfer the entire contents of the file. This transmission of information via the file name was not intended as a normal means of communication between processes and is thus a covert channel.

A covert timing channel consists of a method of communicating information between subjects by altering the use of system resources in order to affect the system response time. One subject (user or process) attempting to communicate with another subject, for example, could either initiate CPU intensive jobs or not initiate them at certain predetermined intervals. The initiation of a CPU intensive job might then represent a 1 while the lack of the job might represent a 0. The subject (in this case most likely a process) which is to receive the message checks the CPU utilization at the specified intervals to detect whether the other has initiated the job or not. In this manner, the two processes could communicate a message one bit at a time. Since there are many other processes that can also affect the CPU utilization, this type of covert channel is often considered less of a threat than the covert storage channels.

3.4 Summary

Many factors go into making a computer system secure. Some of the most basic and important don't involve technology but rather the physical side of the computer. Locks, guards, and surveillance cameras to discourage individuals from gaining unauthorized physical access to computer facilities are probably the most common and easily implemented first line of defense. Established policies and procedures to let people know how to perform specific duties such as properly discarding files, printouts and manuals, or to instruct employees on how to handle visitors or maintenance personnel also help in building a first line of defense. Eventually, especially in today's heavily networked environments, we must turn to the computers themselves to provide security for the data and programs they store and process.

A technique often employed in developing a secure system is the establishment of a security kernel and reference monitor. This involves development of a small subset of the operating system designed specifically to control all subjects (users or processes) access to objects (resources such as files and main memory). While access control is one element of a secure computer system, there are many others which need to also be addressed. These others include Security Models, Authentication, Access

Control, Cryptography, and Auditing—each of which is addressed individually in later chapters. Even after these additional elements of security are considered, other concerns still remain. One such remaining concern is the possible existence of covert channels which are means to transmit information between subjects in violation of the systems established security policy. Covert channels may consist of covert storage or timing channels and are often very difficult to detect.

3.5 Projects

- 3.1 A common practice among computer system vendors is to include a maintenance userid and password on delivered systems. These userid/password pairs are then used to allow vendor service personnel quick, remote access in the case a problem occurs. These userid/password pairs often consist of combinations such as *guest/guest* or *system/maint*. What are the dangers in allowing this type of maintenance or service userid and password to exist on your system?
- 3.2 If the DOS *del* command doesn't actually delete the file, how can you be sure that all remnants of the file you wanted destroyed actually have been? What procedure is necessary to ensure this occurs?
- 3.3 One solution proposed to make systems more secure is to place the operating system functions on a chip (in firmware). What are the advantages and disadvantages to this proposal.
- 3.4 Two examples of covert channels were given in the text. What other techniques could be used to provide covert channels?
- 3.5 How would you test for covert storage channels? What about covert timing channels? Why is testing for them so difficult?

3.6 References

- 3.1 Ames, Stanley Jr.; Gasser, Morrie and Schell, Roger, "Security Kernel Design and Implementation: An Introduction", *Computer*, Vol. 16, No. 7, July 1983, pp. 41-49.

- 3.2 Department Of Energy, “Unix NCSA httpd Vulnerability”, *Computer Incident Advisory Capability Advisory Notice*, F-11, February 14, 1995.
- 3.3 Popek, C.J. and Kline, C.S., “Issues in Kernel Design”, *AFIPS Conference Proceedings*, Vol. 47, 1978, pp. 1079-1086.

3.7 Extended Bibliography

- 3.4 Berson, T.A. and Barksdale, G.L. Jr., “KSOS – Development Methodology for a Secure Operating System”, *AFIPS Conference Proceedings*, Vol. 48, 1979, pp. 365-371.
- 3.5 Gasser, Morrie, Building a Secure Computer System, Van Nostrand Reinhold, New York, New York, 1988.
- 3.6 Hardy, Brian, “An Overview of the Kernelized Secure Operating System (KSOS)”, *Proceedings of the 7th DOD/NBS Computer Security Conference*, Gaithersburg, Maryland, September 1984, pp. 146-160.
- 3.7 National Computer Security Center, *A Guide to Understanding Covert Channel Analysis of Trusted Systems*, NCSC-TG-030, Version 1, November 1993.
- 3.8 Saltzer, J.D. and Schroeder, M.D., “The Protection of Information in Computer Systems”, *Proceedings of the IEEE*, Vol. 63, No. 9, March 1975, pp. 1278-1308.
- 3.9 Schell, R.; Tao, T. and Heckman, H., “Designing the GEMSOS Security Kernel for Security and Performance”, *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, Maryland, September 1985, pp. 108-119.
- 3.10 Schell, Roger, “A Security Kernel for a Multiprocessor Microcomputer”, *Computer*, Vol. 16. No. 7, July 1983, pp. 14-22.

4

SECURITY MODELS

Simply stated, the goal in computer security is to protect the computer system and the data it processes. How successful we are in reaching this goal depends a large part on the amount of care we put into implementing the security controls designed for the system. Before we can implement the controls we need to understand the specific security requirements for our system. The purpose of a ***Security Model*** is to precisely express these security requirements. The model chosen to satisfy the specified requirements should be unambiguous, easy to comprehend, possible to implement, and reflect the policies of the organization. This chapter discusses several security models as well as the Department of Defense's well established criteria for what constitutes a secure computer system.

4.1 Specification and Verification

If the security controls implemented fail, it is usually a result of one of two general types of error: 1) there are errors in the software that control system security, or 2) the definition of what is required for the system to be secure was inaccurate or incomplete. With the exception of certain specific threats such as Trojan Horses and Trap Doors, the first of these two types of errors is addressed through the use of standard Software Engineering practices. The second type of error requires the selection of a security model that meets the requirements of the system and the correct application of this model to the system's development process. If the system has already been built, the job of adding security requires a less formal modeling approach. The formal models addressed in this chapter are intended for use in the design and specification of systems to be built.

For systems requiring the highest degree of security, a formal specification and verification is essential in addition to the formal security model. The goal of the

specification is to describe the proposed behavior of the system in an unambiguous manner that lends itself to verification methods. The purpose of the security verification is to initially prove that the specifications conform to the formal security model. Later, the actual implementation will be tested to verify that it adheres to the specifications. Being able to develop specifications that lend themselves to verification generally requires the use of a special specification language and automated tools to aid in the process. The most popular tools used in security specifications are [4.3]:

- Gypsy Verification Environment (GVE) from the University of Texas.
- Formal Development Methodology (FDM) from the System Development Group of Unisys.
- Hierarchical Development Methodology (HDM) from SRI International.
- AFFIRM developed at the University of Southern California Information Sciences Institute.

These packages not only provide a specification language and their respective tools, they also prescribe an approach to actually designing the system itself. Each of these packages is large and complex and the field of formal specification is far from being mature.

4.2 Security Models

To develop a secure system, a formal model for the security should be included as part of the top-level definition of the system. The purpose of this model is to precisely define the desired behavior of the security-relevant portions of the target system. This desired behavior is strongly influenced by the expected threats to the system and the data it processes. An example can be seen in the differences between the broad security goals found in the military and commercial environments. Military security policy has long been chiefly concerned with disclosure of information to unauthorized individuals. While this is still of concern in the commercial sector, often the disclosure of information is less important than preventing unauthorized modification of information. It is precisely this aspect of security that is addressed when selecting the security model to implement. There have actually been very few security models developed which have received widespread exposure. This is not due to problems in modeling but actually due to the broad applicability of the models that have already been developed. There are a number of principles that lie at the heart of the security

models that will be discussed. How these principles are addressed distinguishes each of the models. The principles to be considered include:

- Identity - can each user, program, object, and resource be uniquely identified?
- Accountability - can users can be held accountable for their actions?
- Monitoring - is a record maintained of user's actions?
- Authorization - do rules exist to govern which entities may access which objects?
- Least privilege - are users restricted to the minimal set of resources needed to perform their job?
- Separation - are the actions of entities prevented from interfering or colluding with the actions of other entities?
- Redundancy - are copies of hardware, software, and data components maintained to ensure consistency, accuracy, and timeliness of results?

4.2.1 Biba

The Biba model is based on a hierarchical framework of integrity levels. The basic elements of the model are:

- S - the set of subjects which are the information processing elements
- O - the set of objects which are passive elements for information storage
- I - the set of integrity levels.
- il - a function used to define the integrity level of a subject or object
- leq - a relation used to define a partial ordering (less than or equal) on the set of integrity levels
- min - a function which returns the greatest lower bound on a set of integrity levels
- o - the capability of a subject to observe an object
- m - the capability of a subject to modify an object
- i - the capability of a subject to invoke an object

The integrity level of an object is assigned based on how catastrophic it would be if it were improperly modified by a subject. Subjects in turn are assigned an integrity

level based on the user itself and on the lowest integrity level required to perform its mission (the least privilege principle described previously). The Biba model actually supports five different integrity policies. The first is the **Low-Water Mark Policy**. In this policy the integrity level of the subject is not static but changes based on its previous behavior. This policy can be formalized in the following manner:

$$il'(s) = \min [il(s), il(o)] \quad (4.1)$$

This axiom states that the integrity level of a subject immediately following an observe access to an object is set to be the lower of the integrity levels for the subject and the object.

The second integrity policy supported by the Biba model is the **Low-Water Mark Policy for Objects**. Similar to the Low-Water Mark Policy, this policy enables the integrity level of modified objects to also change. The Low-Water Mark Policy for Objects adds the following additional axiom for modify accesses by a subject to an object:

$$il'(o) = \min[il(s), il(o)] \quad (4.2)$$

A variation on the Low-Water Mark Policy for Objects is the **Low-Water Mark Integrity Audit Policy** -- the third integrity policy supported by the Biba model. This policy introduces a mechanism for measuring the possible corruption of data. A corruption level (cl) is defined for subjects and objects as follows:

$$cl'(s) = \min[cl(s), cl(o)] \text{ (for each observe access by the subject } s) \quad (4.3)$$

$$cl'(o) = \min[cl(s), cl(o)] \text{ (for each modify access by the subject } s) \quad (4.4)$$

The fourth policy supported by the Biba model is the **Ring Policy**. This policy enforces a strict, unmodifiable integrity level for the life of both subjects and objects. This policy is defined by the following two axioms:

$$s \text{ } m \text{ } o \text{ implies } il(o) \text{ } leq \text{ } il(s) \quad (4.5)$$

$$s1 \text{ } i \text{ } s2 \text{ implies } il(s2) \text{ } leq \text{ } il(s1) \quad (4.6)$$

What the first of these axioms is stating is that if a subject modifies an object then the integrity level of the object must be less than or equal to the integrity level of the subject. Thus, for example, a user would not be able to modify a file which had a 'higher' security classification than the user. The second axiom states that for one subject to invoke another subject, the integrity level of the second must be less than or equal to the first.

The final policy that the Biba model can support is the *Strict Integrity Policy*. This policy consists of the two axioms from the Ring Policy (stating that a subject can't modify an object with a higher integrity level and that a subject can't invoke another subject with a higher integrity level) and adds a third axiom:

$$s \circ o \text{ implies } il(s) \leq il(o) \quad (4.7)$$

This last axiom states that a subject can't observe objects with a higher integrity level. While the Biba model appears to be somewhat flexible in how it can be used to define a security policy for a system, it has not been widely used. It's Strict Integrity Policy, however, is basically the same as what is found in the better known Bell and LaPadula model.

4.2.2 Bell and LaPadula

Probably the best known security model is the Bell and LaPadula Model (BLP) [4.1]. This model is one of a group of *finite-state machine* models. A finite-state machine views the computer as a finite series of states. A transition function determines the next state of the system based on the current state and the value of any inputs which may cause a change in the system. The state is, in essence, defined by the current value of the *state variables* such as the value of the system registers and main memory. The *transition functions* are the various ways that the system can be changed. These functions also form the rules which govern the allowable ways in which the system can change. The security model will consist of a definition of the state variables and transition functions and will start at some provably secure state. Then, if all transitions can be proved to be secure and shown to result in a secure state, the model can, by induction, be shown to be secure.

The BLP model uses the concept of a finite-state machine to formalize its approach. It defines what it takes to be secure, what the allowable transitions are so that an insecure state can not be reached from a secure state, and the various components of the finite-state machine itself. The model is designed to meet the requirements of what is known as the *Military Security Structure* which is

concerned more with the disclosure of information rather than the unauthorized destruction or modification of information. The components defined in the BLP model are:

- Subjects - the system entities which includes the users and processes.
- Objects - the data structures which store the systems information.
- Modes of Access - how a subject interacts with an object.
- Security Levels - follows the military security system in which each object has a classification and each subject a clearance.

The modes of access specified in the model consist of operations such as read, write, and execute, as well as combinations of these basic operations. A state is considered secure if for each triplet consisting of a subject, object, and mode of access, the following properties are satisfied:

- 1) The level of the subject must be at least the level of the object if the mode of access allows the object to be read.
- 2) The level of the object must be at least the level of the subject if the mode of access allows the subject to write.
- 3) The operation may not change the classification of the object.

The first of these properties is commonly referred to as the *simple security property*. It describes what is known as the “no read up” policy. This means that no subject is allowed to read information which is of a higher classification than the subject is cleared for. For example, a user cleared for Secret information may not read information that is classified as Top Secret. The second property is the *confinement property* [4.2, 4.3] but is usually referred to as the **-property* (pronounced “star property”). This property dictates that a subject may not write to an object with a lower classification than the subject has clearance for. This is also known as the “no write down” policy. An example of this second property is a user cleared for Top Secret is not allowed to update a file with a classification of only Secret. These first two principles are illustrated in [Figure 4.1](#). In this figure we show a user cleared for Secret information. The simple security property states this user is not allowed to read information from the Top Secret Domain. The *-property states this user may not write to the Unclassified domain. The user can both read and write to files in the Secret domain. The user may also read files that are unclassified and (based on this property alone) may write to files in the Top Secret Domain.

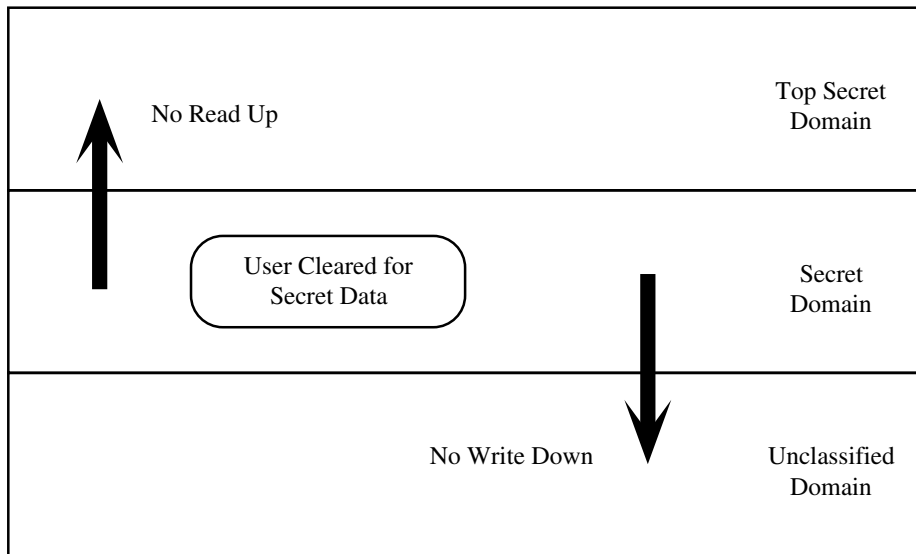


Figure 4.1. Bell and LaPadula Properties.

The third property specified in the BLP model is known as the *tranquillity principle*. Though this last property, which keeps active objects from having their classification modified by an operation, is not strictly required for military security structures, Bell and LaPadula adopted it as an additional security measure. Essentially, this last property states that in terms of classification or clearance the subjects and objects found in each of the domains illustrated in [Figure 4.1](#) will not change. The only way the domains will change is with the creation or deletion of subjects or objects.

The rigid adherence to the *-property was recognized by Bell and LaPadula as more strict than military security requires. Consequently, the idea of *trusted subjects* was developed. A trusted subject is one which can be relied on to not compromise information even if some of its invoked operations violate the *-property. Strict adherence to the *-property is only required of untrusted subjects. A criticism of the BLP model as described is the 'flat' way it handles security levels.

The BLP model, or the BLP model along with some modifications, has been used in the design and implementation of several systems including MULTICS for the Air Force Data Services Center, the Kernelized Secure Operating System (KSOS), and the Honeywell Secure Communications Processor (SCOMP) [4.5]. The principal problem with the model has not been with what it allowed, but with what it disallows. The model has been criticized most often for being too restrictive and too concerned with the

military security policy which may not be applicable in all environments. Other models have been developed to address these other environments.

4.2.3 Clark-Wilson

The BLP model addresses the military security policy's goal of unauthorized disclosure and declassification of information. In many environments, however, disclosure is less important than preventing the unauthorized modification of data. An example of this is the majority of commercial data processing that deals with management and accounting of financial assets. In this environment preventing fraud is often more important than disclosure of information. This switches the emphasis from that of a privacy issue to one of enforcing integrity. The Clark-Wilson model was developed to address this commercial environment [4.7].

Besides stating that integrity policies are often of more importance than disclosure policies, Clark and Wilson also assert that there is a distinct set of policies that can be used to address the integrity issue and that separate mechanisms are required to do so. Clark and Wilson state the goal of a commercial system concerned with integrity of data is to insure that no user can modify data in a manner that would result in the loss or corruption of assets or accounting records for the company. This is true not only for unauthorized individuals but also individuals normally authorized access to the information. The model uses two categories of mechanisms to realize this goal. The first is called *well-formed transactions* and the second is *separation of duty* among employees.

The purpose of a well-formed transaction is to ensure that a user can't alter data arbitrarily. Instead, data can only be altered in specified ways that severely constrain the possible changes in order to preserve its internal consistency. An example of a well-formed transaction mechanism is an audit log which records all data transactions. This provides system managers the ability to later recreate not only the sequence of events which may have led to an unauthorized modification but the original data as well. Auditing, as a well-formed transaction mechanism, is analogous to bookkeepers who, before the advent of computers, were once instructed to maintain their books in ink. When an error occurred, the bookkeeper also made the correction in ink. If an erasure was ever encountered, it was an indication of fraud. This method served as a deterrent to fraud, not as a mechanism that made it impossible to perform. The same is true of audit logs. They don't make it impossible to perform unauthorized modifications of data but instead simply deter it by making it easier to detect when fraud occurs.

Another example of a well-formed transaction mechanism is found in accounting systems which base their transactions on a double-entry bookkeeping principle. This principle requires that any modification to accounting information must be accomplished in two parts. If a withdrawal from a cash account is made, for example,

an appropriate entry must also appear in another location such as an accounts payable account. If the transaction does not appear in both locations it will be detected when the books are balanced. Once again, this does not make it impossible for an unauthorized action to occur but rather deters it from happening.

Applied to computer systems, well-formed transaction mechanisms result in the identification of only certain programs which may modify data. These programs will thus perform very specific, and often limited, transactions. The programs themselves must be inspected and tested to ensure they have been written correctly and perform only their intended purpose. Their installation and modification must also be tightly controlled to ensure the integrity of the programs themselves.

The second category of mechanism used to reach the goal of commercial security is the separation of duty. This category attempts to maintain consistency of data objects by separating all operations into several parts and requiring that each part be performed by a different subject (e.g., a user). Basically this means that any user who is allowed to initiate a well-formed transaction is not allowed to execute it. For example, the process of purchasing some product might involve several steps. The first step might be the authorizing of the purchase. The second the ordering of the item. The third step might entail receiving and signing for the item and the last step could be the authorization of payment for the item. If each step in this process is performed by a different individual, the chance of collusion among employees for the purpose of fraud is reduced. To implement this mechanism on a computer system would entail permitting users to run only certain, specific programs. One individual, for example, would not be allowed to run the programs that governed the authorization, ordering, recording of reception, and accounting programs for the purchase of products as illustrated in our previous example.

There are several differences between the controls used for military security found in models such as that described by the Bell and LaPadula model and the controls for commercial security described in the Clark-Wilson model. The first is in the way in which data is treated. In military security models data is assigned a security level and the mode of access granted depends on this level and the level of the subject. In the Clark-Wilson model data is associated not with a security level but rather with a set of programs which are allowed to access and manipulate it. Secondly, the subjects, or users, are not given authority to access certain data or classification of data but instead are given permission to access, or execute certain programs which in turn access specific data. Put simply, in military security users are constrained by the data they can access while in the Clark-Wilson model they are constrained by the programs they can execute. While military security policies use the triplet *subject/object/mode of access* to determine access restrictions, the Clark-Wilson model uses *subject/object/program* where the mode of access, and thus the operation to be performed on the data, is implied by which program is being executed.

The Clark-Wilson model defines four elements to describe the operation of the model. Constrained Data Items (CDIs) are those data items in the system for which the integrity model must be applied. Unconstrained Data Items (UDIs) are those items which are not covered by the integrity model and may be manipulated freely. Transformation Procedures (TPs) change a set of CDIs which may also include new information in the form of a UDI from one valid state to another. Integrity Verification Procedures (IVPs) are used to confirm the internal data consistency of CDIs. Figure 4.2 illustrates the operation of these elements.

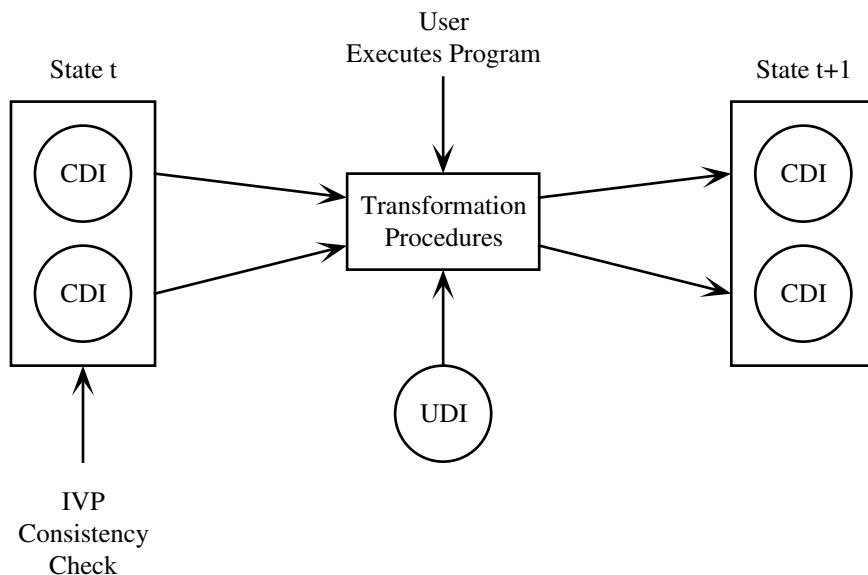


Figure 4.2. Clark-Wilson integrity model.

In this example, a user executes a program which will, based on the TP, transform two CDIs at State t and a UDI into the two CDIs found at State $t+1$. The IVP verified the consistency of the CDIs at State t before the transformation took place, and the user's authorization to execute this program (and transformation) would have been authenticated. In this example, the UDI is also supplied as input from the user. Implementations of the Clark-Wilson model have been developed independently by both Lee [4.6] and Shockley [4.7]. These implementations have shown that strong data typing is important and a transaction-oriented approach works well with the integrity based approach of the Clark-Wilson model.

4.2.4 Goguen-Meseguer

The Goguen-Meseguer model introduces the concept of noninterference between users [4.4]. In essence this model prevents a higher-level user from interfering with a lower-level user. Simply stated, a subject, user, or group of users is said to be noninterfering with another subject, user, or group of users if what the first does has no effect on what the second can see. The elements of this model are:

- The set of all possible states for the system.
- The set of users.
- The set of commands which the users can execute.
- The set of outputs.
- The next state.
- The set of security levels.

In addition, there are two basic definitions:

- 1) A user is said to be *noninterfering* with a second user if, for any two strings consisting of a sequence of commands in which the commands issued by the second user are the same, the outputs for the second user are the same.
- 2) Given an initial secure state, the system is *multi-level secure* if the first user is noninterfering with the second user, unless the level of the first is less than or equal to the second user.

To illustrate this point, consider the following multi-level secure example. Suppose we have a user cleared for Top Secret information who wants to use an object classified at the Unclassified level to modify an object at the Secret level. In the BLP model this would not be allowed since the Top Secret user is sending information to the Unclassified level, in this case to modify some other data. The Unclassified level changing data at a higher level is allowed, just not the Top Secret user communicating with an Unclassified object. This violates the “no write down” *-property. While there is a possibility of a covert channel if we allow this operation, the essence of this change is nothing more than an upward information flow which is allowed in the BLP model. This operation would not, however, necessarily violate the rules associated with the Goguen-Meseguer model because, according to the definitions, the system could still be multi-level secure as long as the higher-level users actions did not interfere with lower-level users.

Implementations of the Goguen-Meseguer model have focused on the idea of separating users based on their protection domains to avoid interference between groups.

A version of this model was used by Honeywell in the Secure Ada Target research project. This implementation followed an object oriented, capability based approach which restricted performance of actions on objects to subjects that had access to the domain in which the object resided.

4.3 The Trusted Computer System Evaluation Criteria

As has been discussed, security models can aid in the development of security requirements for computer systems. Other documents have also been developed which aid in this process. The Department of Defense's (DoD) Trusted Computer System Evaluation Criteria (TCSEC), commonly referred to as the **Orange Book** because of the color of its cover, provides the DoD a basis for specifying security requirements in a computer system [4.2]. It also serves as an example of an approach to define specific security criteria for a chosen security model. Though officially only a standard for the DoD, this document has served as a basis from which to evaluate various computer security features for a number of years. It is not uncommon to see products developed for the commercial sector proclaim adherence to specific Orange Book requirements in their advertisements because of the widespread understanding of the basic requirements found in the book. The book thus provides a standard to manufacturers for what is required to satisfy basic requirements for a trusted system. Note that while the criteria are discussed in great detail in Chapter 16, *Security Standards*, they are relevant to this discussion and thus, presented here in brief.

The criteria divides systems into four divisions based on the requirements they satisfy. The divisions are subdivided into classes to further categorize the evaluated systems. At the bottom is Division D which has only one class. This division is reserved for those systems that have been evaluated but have been found to not meet the requirements for any of the higher classes. The next level up is Division C which has two classes. This level introduces discretionary access controls. The next level is Division B which consists of three classes and which introduces the concept of mandatory access controls. Finally at the top is Division A which also has only one class. Systems assigned this topmost rating are functionally equivalent to the highest class in the previous division, B, but have had much more extensive formal design specification and verification techniques applied.

Each of the classes, except Class D, specify requirements in four broad areas: **Security Policy**, **Accountability**, **Assurance**, and **Documentation**. The **Security Policy** describes the set of rules that govern whether a given subject can access a specific object. **Accountability** includes such functions as identification, authentication, and auditing. **Assurance** requirements provide criteria from which to

evaluate the system's adherence to the other requirements. It includes concerns such as covert channel analysis, trusted recovery, security testing, and continuous protection (the mechanisms that enforce security must be protected themselves). Finally, Documentation requirements specify the documents required for each class and the basic contents of each document. The types of documents that may be required include Security Features User's Guide, Trusted Facility Manual, Test Documentation, and Design Documentation. All of these requirements together make up what the Orange Book refers to as the **Trusted Computing Base (TCB)**. The TCB includes all hardware, software, and firmware which are used to enforce the systems security policy.

While the Orange Book describes four different divisions of trusted computer systems, it really splits them into only two categories. The first are those systems which enforce only **Discretionary Access Controls** which limits processing to a single level of security. The second is **Mandatory Access Controls** which allows processing of information from several different security levels on the same machine. This is known as *multi-level security*.

4.3.1 Discretionary Access Requirements

Discretionary Access Control requirements can be thought of as separating users from information on a need-to-know basis. This means that all individuals who use the system are cleared for the security level of the data with the highest classification processed on the system. Even though they are cleared at that level, they may not actually have a need to know what the data is. This is in keeping with the military's general rule of thumb which keeps access to classified data to as few people as possible. If a problem occurs and access is granted to a user for a document which the user does not have a need to know, the security breach is considered minimal since the user was still cleared for that level of information.

The requirements for providing discretionary security include a protected mechanism such as passwords to authenticate the identity of the user. In addition, the authentication data must itself be protected. An example of this would be encrypting the password file so that, should an individual obtain a copy of it, the users and their passwords would not be readable. Discretionary security also requires that access between subjects (users) and objects (files and programs) be controlled in a manner that allows sharing of the objects among subjects in specified groups while preventing access to subjects outside of the group.

At the C2 level, *object reuse* is introduced as a requirement of discretionary security. Object reuse entails the clearing of any portion of a storage device or memory location that has been previously used before access to it is granted. This ensures that a previous subject's information which was once stored on the device or in memory can

not be obtained by a subject. This even includes encrypted representations of information.

The C2 level also introduces the idea of *audit data*. The purpose of audit data is to provide a recorded trail of accesses to the objects protected by the system. This audit trail should include events such as the use of identification and authentication mechanisms, the access of objects, deletion of objects, and actions taken by computer operators and system and security administrators [4.2]. For each of these events the date and time of the event, user, type of event, and whether the event was successful or not should be recorded. The Orange Book also requires that the system administrator be able to selectively audit the actions of any specific individual or groups of users based on individual user identity. In addition to these requirements, the audit trail itself should be protected so that access to it is restricted not only to prevent unauthorized deletion but to limit read access as well. Other requirements for discretionary security include varying levels of assurance and documentation, depending on the specific class.

4.3.2 Mandatory Access Requirements

Mandatory access controls are required to implement multi-level security. In this division, Division B of the Orange Book, the requirements are designed to maintain the separation between different security levels of subjects and objects. Since unauthorized disclosure of information to individuals can result in prosecution, the controls for these systems can be said to be ‘mandated.’ For mandatory access, none of the requirements levied at the lower discretionary access level are eliminated. Instead, additional requirements are added.

For the system to be able to maintain the separation between the different levels of subjects and objects, a method needs to be implemented which identifies the respective clearance and classification levels. The Orange Book introduces the concept of *Sensitivity Labels* at the B1 Class in order to satisfy this requirement. Sensitivity labels are associated with each subject and object under control of the system. Whenever new data or new users are added to the system, the TCB is required to request and assign a security level for them. Additional requirements associated with labels include the maintenance of the label when objects are exported to other devices, and the need to include the sensitivity label as part of each page of hardcopy output. The labels mandated by the Orange Book are required to handle not only a minimum of two different hierarchical security levels but also any number of non-hierarchical categories. At the higher classes in this division additional requirements include the assignment of minimum and maximum security levels to actual physical devices (such as printers and terminals) to match constraints imposed by the physical environment.

The security model used by the Orange Book to specify its mandatory access control requirements is a version of the Bell and LaPadula model. The controls required

by the Orange book for this multi-level security system state that a subject can read an object only if the hierarchical sensitivity level of the subject is greater than or equal to that of the hierarchical level of the object to be read. In addition, the subject's non-hierarchical sensitivity categories must include all of the non-hierarchical categories of the object. A subject that wishes to write to an object can do so only if the subject's hierarchical sensitivity level is less than or equal to the hierarchical sensitivity level of the object and the subject's non-hierarchical sensitivity categories include all of the non-hierarchical categories of the object.

For obvious reasons, the integrity of the labels used by the system is of utmost importance. Consequently other requirements levied on systems starting at Class B1 include the addition of the object's sensitivity level on audit records and the auditing of any attempt to override security labels. At Class B2 covert channel analysis is also added to determine if any communication channel exists which might allow the transfer of information in a manner that would violate the security policies set.

4.4 Summary

How successful our security measures for a computer system are depends a great deal on how much care is used in implementing the chosen security controls. Prior to implementing the security controls, we need to select a security model which precisely expresses the system's security requirements. This model should have four characteristics. It should be unambiguous, possible to implement, easy to comprehend, and should be in keeping with the security policies of the organization.

One of the first models, the Biba model, allows for a variety of integrity policies. The most widely known model in use today, however, is the Bell and LaPadula model. This model implements military security policy which is chiefly concerned with disclosure of information. The Clark-Wilson model, in contrast, addresses a major commercial concern of protecting the integrity of data. A third model, the Goguen-Meseguer model, introduces the concept of noninterference between users.

The Department of Defense's Trusted Computer System Evaluation Criteria (TCSEC) provides requirements and specific criteria to develop a system which implements a chosen security model. The model chosen for the TCSEC is a version of the Bell and LaPadula model. While it uses four divisions to classify the level of security implemented in a system, the TCSEC chiefly describes requirements for two specific types of controls; discretionary access controls and mandatory access controls. Discretionary controls separate users and information on a 'need-to-know' basis while mandatory access controls separate them based on different security classifications.

4.5 Projects

- 4.1 The Clark-Wilson model describes the auditing of transactions as a well-formed transaction mechanism. How does recording details relating to the modification of data ensure data integrity and internal consistency?
- 4.2 Recall the example given for the Goguen-Meseguer Model where a Top Secret user accesses an object at the Unclassified level to modify an object at the Secret level. Describe how this situation can lead to a possible covert channel.
- 4.3 Why does the TCSEC not require Security Labels for discretionary access controls? What other measures might be used to separate information of this type?
- 4.4 The Bell and LaPadula model allows users to write to information at a higher level of classification than they are cleared for. Besides the obvious possibility of data integrity problems, what concerns are there with allowing this type of transaction to occur?
- 4.5 Since Trojan Horses and Trap Doors are not ‘bugs’ in the security software that can be discovered using normal testing procedures, what methods could be used to detect these types of malicious software? Is prevention or detection easier? Explain.

4.6 References

- 4.1 Bell, D.E. and LaPadula, J., *Secure Computer Systems: Mathematical Foundations and Model*, Technical Report, M74-244, MITRE Corp., Bedford, Massachusetts, October 1974.
- 4.2 DoD 5200.28.STD, *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985.
- 4.3 Gasser, Morrie, Building a Secure Computer System, Van Nostrand Reinhold, New York, New York, 1988.

- 4.4 Haigh, J.T., "A Comparison of Formal Security Policy Models", *Proceedings of the 7th DOD/NBS Computer Security Conference*, Gaithersburg, Maryland, September 1984, pp. 88-111.
- 4.5 Landwehr, Carl E., "Formal Models for Computer Security", *ACM Computing Surveys*, Vol. 13, No. 3, September 1981, pp. 247-278.
- 4.6 Lee, T.M.P., "Using Mandatory Integrity to Enforce 'Commercial' Security", *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, Oakland, California, April 1988, pp. 140-146.
- 4.7 Shockley, W. R., "Implementing the Clark/Wilson Integrity Policy Using Current Technology", *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988, pp. 29-37.

4.7 Extended Bibliography

- 4.8 Bell, D.E., and LaPadula, J., *Secure Computer Systems: Unified Exposition and Multics Interpretation*, Technical Report, MTR-2997, MITRE Corp., Bedford, Massachusetts, July 1975.
- 4.9 Biba, K. J., *Integrity Considerations for Secure Computer Systems*, Technical Report, MTR-3153, MITRE Corp., Bedford, Massachusetts, April 1977.
- 4.10 Clark, David D., and Wilson, David R., "A Comparison of Commercial and Military Computer Security Policies", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 184-194.
- 4.11 Gove, Ronald A., "Extending the Bell & LaPadula Security Models", *Proceedings of the 7th DOD/NBS Computer Security Conference*, Gaithersburg, Maryland, September 1984, pp. 112-119.
- 4.12 Keefe, T.F.; Tsai, W.T. and Thuraisingham, M.B., "A Multilevel Security Model for Object-Oriented Systems", *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988, pp.1-9.

- 4.13 Korelsky, Tanya; Dean, Bill; Eichenlaub, Carl; Hook, James; Klapper, Carl; Lam, Marcos; McCullough, Daryl; Brooke-McFarland, Clay; Pottinger, Garrel; Rambow, Owen; Rosenthal, David; Seldin, Jonathan and Weber, D.G., "ULYSSES: A Computer-Security Modeling Environment", *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988, pp. 20-28.
- 4.14 Mayfield, Terry; Roskos, J. Eric; Welke, Stephen; Boone, John and McDonald, Catherine, "Integrity in Automated Information Systems", Technical Report 79-91, Library No. S-237254, National Computer Security Center, September 1991.
- 4.15 McLean, John, "Reasoning About Security Models", *Proceedings of the 1987 Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 123-131.
- 4.16 Page, John; Heaney, Jody; Adkins, Marc and Dolsen, Gary, "Evaluation of Security Model Rule Bases", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 98-111.
- 4.17 Pittelli, Paul A., "The Bell-LaPadula Computer Security Model Represented as a Special Case of the Harrison-Ruzzo-Ullman Model", *Proceedings of the 10th National Computer Security Conference*, Gaithersburg, Maryland, September 1987, pp. 118-121.
- 4.18 Roskos, J. Eric; Welke, Stephen; Boone, John and Mayfield, Terry, "A Taxonomy of Integrity Models, Implementations and Mechanisms", *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 541-551.
- 4.19 Saydjari, Sami, "A Standard Notation, in Computer Security Models", *Proceedings of the 9th National Conference on Computer Security*, Gaithersburg, Maryland, September 1986, pp. 194-203.
- 4.20 Summers, Rita, Secure Computing, McGraw-Hill, New York, NY, 1997.
- 4.21 Taylor, Tad, "Formal Models, Bell and LaPadula, and Gypsy", *Proceedings of the 10th National Conference on Computer Security*, Washington, DC, September 1987, pp. 193-200.

5

USER AUTHENTICATION

The first line of defense a computer system has against intruders is user authentication. The user authentication system attempts to prevent unauthorized use by requiring users to validate their authorization to access the system. This validation is often accomplished with the use of a password that must be presented to the system. Other authentication schemes require the user to present a physical key or take advantage of the uniqueness of a user's physical attributes such as fingerprints. This chapter discusses various authentication schemes and examines their implementations, advantages, and drawbacks.

5.1 Authentication Objectives

The primary objective of an authentication system is to prevent unauthorized users from gaining access to a private computer system [5.8, 5.15]. Much like a normal door lock authorized users are given a key to the system thus keeping unauthorized users out. This does not mean, however, that unauthorized users are unable to gain access to the system. It has become quite common for authorized users to be lax in the protection of their access key. That is, unauthorized users are able to access the system by using another person's key and appearing to be an authorized user. This is possible because the system does not authenticate the identity of a user, only who the key holder claims to be [5.15]. Since the authentication system can not verify the user's true identity, methods must be in place to reduce the opportunity for an unauthorized user to appear as an authorized user and access the system. This is accomplished with one or more of the numerous authentication methods.

5.2 Authentication Methods

A computer system may employ three different types of authentication methods to prevent unauthorized users from gaining access [5.9, 5.19]. The first, and most common, method is through the use of informational keys. A system that uses informational keys requires the user to provide specific information to access the system. Examples of informational keys are passwords, pass phrases, and questionnaires. The second type of key is a physical key or Token. Physical keys are objects that a user must have to access the system. Examples of physical keys include magnetic cards, smartcards, and security calculators. The third type of keys are biometric keys. Biometric authentication systems rely on a user's physical attributes to grant or deny access. Examples of biometric keys are fingerprints, voice prints, retinal prints, and hand geometry. Each of these types of keys are discussed below.

5.2.1 Informational Keys

Informational keys are words, phrases, or question responses that an authorized user knows and can provide to the system when challenged. The passwords and phrases may either be user selected or assigned by the system. While the questions used in questionnaire authentication may be provided by the user (along with the answer), they are typically selected by the system.

5.2.1.1 Passwords

One of the most commonly used authentication schemes employs passwords [5.19]. Along with a user name, a password must be presented to the authentication system to gain access. The system grants access to a user only when the password for the specified user matches the password on record. This guarantees that only those users that have the correct password for a specific user account can access the computer.

Many systems allow the user to create their own password so that it is more easily remembered. User selected passwords, however, are often easily guessed [5.9, 5.15]. Examples of easily guessed passwords are names and words found in dictionaries, special dates, family or pet names, or words associated with a person's job or company [5.7, 5.9]. Passwords that are generally accepted have at least six characters (preferably eight), at least one of which is a number, symbol or punctuation mark [5.15]. In general, a user's password should be both easy to remember but difficult to guess. A good method for selecting a strong password is to use the first, or last, letters from each word in a memorable phrase and then mix in some numbers or punctuation. For example, starting the phrase "I will never forget to

wash behind my ears” and producing “Iwnf2wbme!” as the password. This often results in a relatively strong password that is difficult to guess but easy to remember.

Even though a person may select a password that is difficult to guess, his or her actions may lead to its compromise. A user can severely reduce the chance of their password becoming public knowledge by following three rules: (1) never write a password down, (2) never tell anyone the password, and (3) change the password on a regular basis. A large number of people select a very complex password and write it down because it is too difficult to remember [5.7]. This may seem reasonable, however, it has proven to be risky. Writing a password down—even if it is kept in a wallet—allows both the casual observer and co-workers easy access to what would normally be considered a secure computer account. Just as risky as writing a password down is revealing a password to another person. Neither the actions of another person nor an unknown observer can be guaranteed beneficial to the user. To prevent unauthorized users from accessing a computer account, authorized users should change their password on a regular basis. This will reduce the possibility of intruders, both known and unknown, from unauthorized access. A good rule of thumb is to treat an account passwords with the same respect one treats a Bank ATM card password—never share it, give it out, or write it down.

An extension of the concept of changing a password on a regular basis is the one-time password. A one-time password changes with each use [5.8]. That is, every time a user correctly presents their password to the authentication system, it changes. This requires an unauthorized user to obtain the account name and associated current password to gain access. This prevents an unauthorized user from using old passwords that may have become public knowledge. In the event an unauthorized user does obtain and use the current password, their presence is noticed when the authorized user attempts to log in with a password that has already been used.

To implement the single use password scheme two issues must be addressed: (1) distribution of the password list and (2) protecting the password list from theft. Distribution of the password list may either occur all at once or as each new password is needed [5.7]. The advantage of distributing the new password each time the authorized user logs in is the elimination of the need for a separate list that may be easily lost, stolen, or copied. The disadvantages of this approach, however, are numerous: the unauthorized user would not only have the proper password for the one time it was valid, but will be provided the new password, the password list or algorithm residing on the computer may be compromised, or the electronic distribution of the password to the user may be monitored. The other alternative, distribution of the entire password list at one time, eliminates these problems, but creates others. Distribution of the entire list at one time requires the authorized users to protect the distribution only once. The delivered list, however, must constantly be protected from theft, duplication, and loss. In the event that the list is stolen, copied, or lost, the computer system becomes vulnerable and a new list must be generated, installed, and distributed. A good

compromise to these two approaches has been developed in the form of physical keys, specifically the calculator and the smart card.

5.2.1.2 Questionnaires

Questionnaire authentication attempts to validate a user with questions that an intruder is unlikely to know [5.7]. The system asks questions such as the name of the user's first pet, the color of the user's parent's home, or the name of the user's favorite teacher. For this system to function properly the user must prepare the questionnaire when the account is established and periodically update it to incorporate new information. In some cases the user is even asked to write the questions so that the requested information is not consistent across an organization. Questionnaire authentication, however, is not likely to be used in a high security environment [5.7]. Since much of the information used in this process can be researched by an intruder, questionnaire protection is often insufficient and thus rarely used.

5.2.2 Physical Keys

Physical keys are items that a user must have in their possession to access the system [5.19]. Much like a key is required to enter a locked room, a special key may be required to access a computer system or network. Along with a user name or password, the user must present, or insert, their personal key to gain access. In the event that the key and identification number do not match or are invalid, the user is denied access.

Three commonly used physical keys are magnetic cards, smartcards, and specialized calculators. These keys are widely used because they provide a higher level of security than passwords alone, are simple to use, and are relatively inobtrusive. The greatest problem with physical keys arises when they are lost or broken making authorized access to the system impossible until key is replaced.

5.2.2.1 Magnetic Cards

Most people recognize the common credit card to be a magnetic card. A magnetic card is a piece of non-conductive material with an additional piece of magnetic recording material attached to it [5.1, 5.19]. Physical standards for magnetic cards have been developed by The American National Standards Institute (ANSI) and the International Standards Organization (ISO) [5.1]. These standards allow multiple industries to take advantage of the technology.

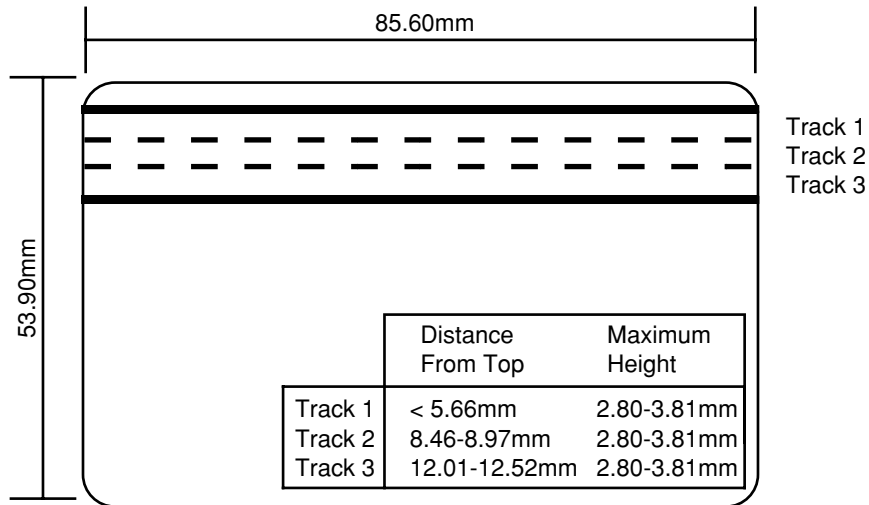


Figure 5.1. The layout and design of a magnetic card.

The specific design of the magnetic card as described in ANSI/ISO standard 7810-1985 is provided in [Figure 5.1](#) [5.1]. These standards indicate that the card must measure 85.60 mm wide by 53.98 mm high with a thickness of .76 mm. The magnetic strip on the backside of the card covers the full width of the card. The exact height and placement of the magnetic strip are not as important as the three data tracks within the strip. Each of the three tracks in the strip must measure between 2.80 mm and 3.81 mm high. The first track can not be more than 5.66 mm from the top of the card. The second track should lay between 8.46 mm and 8.97 mm from the top and the third track should lay between 12.01 mm and 12.52 mm from the top.

To take advantage of the magnetic card technology, the authentication system must have card readers attached to each terminal or physical access path such as a door. A user inserts their magnetic card into the reader and if authorized, granted access to the system. Some magnetic card systems also require each user to enter a personal access number to verify that the card holder has not simply found the card, but is the person that was assigned the card. When the access number and magnetic card have been verified to be a valid matching pair, the user is authenticated and may access the system. The validation process occurs in four stages:

- 1) Input. The process begins when a user enters a card and possibly an access number.
- 2) Encryption. If the user was required to enter an access number, that number along with the card provide identification as a means

of determining authentication. Together, the card and the access number will be used to look up a comparison value in the system and generate an encrypted password. If the user was not prompted for an access number, the user's identity cannot be verified and the card simply provides an encrypted password for comparison with a value stored in the system.

- 3) Comparison. If there the user provided password and the system password match, the user will be given access.
- 4) Logging. Regardless of the outcome of the comparison, the attempt to access the system is logged. These logs are used to locate unusual behavior such as excessive amounts of failed access attempts and concurrent accesses at different locations.

5.2.2.2 Smartcards

The term “smartcard” has come to mean one of many types of credit card sized devices that authenticate users. In 1990 the ANSI and the ISO approved a standard for the original “smart card”; an identification card that contain integrated circuits (ANSI/ISO standard 7816/1&2) [5.2]. Like magnetic cards, these smartcards may contain information about the identity of the card holder and are used in a similar manner. This smartcard, however, has the ability to perform computations that may improve the security of the card and impede its unauthorized alteration or duplication [5.15, 5.21]. For example, the card may be programmed with the capability to apply its own encryption to the data, validate the integrity of the data, or verify the authenticity of the card [5.7, 5.15]. The issuing organization must decide what calculations a card will perform when used.

Other types of cards that have adopted the smartcard name include proximity cards and one-time password generating cards, or devices. Proximity cards are credit card-sized devices that, when placed in the vicinity (or proximity) of a sensing device, provide access or authentication. These devices are typically used only for physical access, such as to a secure lab or parking lot. The one-time password generating devices are not only available in the convenient credit card sized package, but are also available as key-fobs, and software-based packages. This is possible because they are rather simple devices. They are initialized with a distinct value. This value is then passed through a special algorithm to generate a new number. This number is then used as the input for the next calculation, which is the same as the first. This process continues automatically at predetermined intervals. These intervals may be based on time (e.g., every 30 seconds) or usage (e.g., immediately after it is used). This number by itself is meaningless. When combined with a user name, additional pass code, and authentication device, it provides a very robust authentication system. Typically, a

user will provide this calculated number, along with a constant personal pass code, as their password to a system. The authentication device, or server, performs the same calculations to generate a comparison password. If the two passwords agree, the user is authenticated and granted access. Because access passwords change frequently, password cracking becomes very difficult, if not impossible. This process is similar to that used with password calculators, the primary difference being the need for user interaction with a password calculator device.

5.2.2.3 Calculators

Another type of physical security key is the security calculator. A security calculator is a small device that looks very much like a simple calculator with the addition of a few specialized functions [5.7, 5.15]. The extra functions perform the computations necessary to authenticate a user on a computer system. In addition to possessing a personalized calculator, users are also required to remember a personal access code to obtain access to a system.

One common authentication process begins when a user presents their user name to access a computer system. The system responds with a challenge value. The user must enter their personal access number along with the challenge into their calculator. The calculator performs a series of mathematical computations on the challenge value and computes a response value. The user must present the response value to the system. The system will grant the user access if the number presented matches the value expected by the system. The response value from the calculator will be incorrect if the personal access number is incorrect, the user does not use their personalized calculator, or the challenge number is incorrectly entered.

Other authentication schemes that employ a calculator exist. These schemes work in a similar fashion—they present a challenge value to the user and require a specific response that only a calculator with the proper access code can correctly produce. These schemes, like the one presented above, provide numerous advantages over common password schemes while only incurring minimal drawbacks.

The benefits of using calculator authentication are quickly observed upon its implementation. Calculator authentication requires no additional hardware to be attached to the computer system. Thus, changing from a password based system to a calculator based system requires only minor software modifications. The only new hardware introduced during the installation process is an authentication server and the calculators. Additionally, calculator replacement is very inexpensive, often costing less than \$10 each. The only other type of key that is as secure and inexpensive to distribute is the biometric key.

5.2.3 Biometric Keys

Recent advancements in technology have made it possible for biometrics to be used in the user authorization process. Biometrics are the measurements of personal characteristics such as a voice tonality or fingerprint patterns [5.15]. Biometric keys provide many advantages over informational or physical keys [5.15]. The three primary advantages of biometric keys are: (1) they are unique, (2) they are difficult to duplicate or forge, and (3) they are always in the possession of their authorized user. These characteristics make biometrics an excellent method for user authorization. Commonly used biometric keys include voice prints, fingerprints, retinal prints, hand geometry, and signature analysis [5.15, 5.19].

5.2.3.1 Voice Prints

Voice prints are mathematical representations of a person's speech patterns [5.3, 5.14]. These representations are often used by computer and telephone voice navigation systems. These systems, for example, respond to a user's audible request to speak with an operator or open a file. User authentication systems function in a similar manner, however, they can also attempt to determine the identity of the speaker. This is accomplished by obtaining an unverified identity and speech sample from a user and comparing it to a previously authenticated speech sample. Before the voice print authentication process can occur, however, the system must be initialized with authentic user speech samples.

During the installation process each user will be required to provide a speech sample for authentication. The sample may consist of either complete phrases or single, unrelated words. While single word identification has been shown to be slightly more accurate than phrase identification, it has also been shown to be more susceptible to impersonation [5.3]. It also has a greater chance of incorrectly identifying the speaker [5.23]. Once the format of the speech has been determined, the actual words must be chosen [5.3]. After a user makes a speech sample, they will use the voice authentication system to gain access.

The complete voice print authentication process is accomplished in four steps: (1) pre-processing, (2) sampling, (3) testing, and (4) system response [5.22].

In the first step, pre-processing, a user wishing access identifies his or herself to the system. This indicates to the system whom the user claims to be and which authentic sample the system should use for comparison.

In the second step, sampling, the user speaks a predetermined statement into a microphone connected to the system. In the event the system cannot interpret the speech sample, the system requests the user to speak again. Besides poor enunciation, two problems that may require the user to speak again are background noise and unusually fast or slow speech. Most background noise, however, can be filtered out

electronically or with physical devices such as barriers or microphone padding. Like background noise, most timing problems are also easily resolved electronically. When there are no significant problems and the system can accurately interpret the speech, the testing process begins.

The third step, testing, uses the identity information and the voice sample to determine whether the user will be granted access to the system. The system uses the identity information to select the voice print that will be the basis for identification of the user. Characteristics of the speech in the two samples, such as the acoustic strength and frequency, are compared and weighted according to their importance in matching between the samples. These values are totaled and the result used to determine whether the user is granted access.

The final step of the process occurs after the comparison of the two voice prints, when the authentication system either grants or denies the user access. In either case, the system logs the time and date that access was attempted.

The log files produced by the system are useful in locating and resolving potential problems such as unclear authentication samples and voice print forgeries. The logs will reveal when a user continuously has a difficult time authenticating themselves. This may indicate that either a user's voice has changed or that someone is attempting to subvert the system. System attacks, as well as voice print forgery, will also stand out when a user has been simultaneously authenticated at different locations. While voice prints are unique, advancements in digital recording technology make it possible to record a near perfect voice print. This may be of little concern on site where users are watched as they login. Remote system attacks, however, present a greater problem.

Remote system access, while useful, presents numerous problems to the voice print authentication system. The first problem concerns how a user accesses the system. That is, how does a remote user present their voice print for authorization. Creative solutions involving multiple phone lines or authorization before a connect signal can be developed. These solutions, however, are often cumbersome and complicated. The second problem with remote access concerns the authenticity of the live voice print. It is difficult to determine whether the authentication comes from the authorized user or from an unauthorized user with a high quality or digitally generated recording. For these reasons, remote access is often limited or is not available. These remote access problems are not limited to voice print authentication, but are common to most biometric authentication methods.

5.2.3.2 Fingerprint

Fingerprint authentication is the processes of granting or denying access to a user based on their fingerprint [5.6, 5.15, 5.17]. This is accomplished by obtaining a

fingerprint sample from a user and comparing it to a previously authenticated fingerprint. Before the fingerprint authentication process can occur, however, the system must be installed. This includes installation of fingerprint readers and the recording of each user's fingerprint.

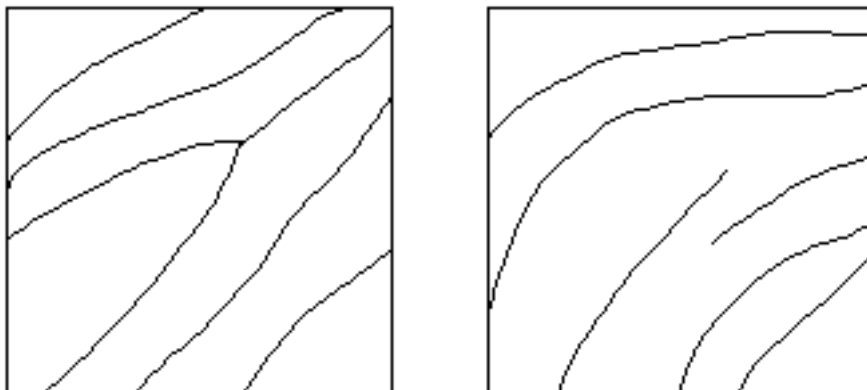
In addition to installing the proper hardware at key locations, each user must have a series of fingerprints recorded as a basis for future system access. Multiple prints of multiple fingers are recorded so that personal injury does not affect a user's ability to access the system.

Once the prints are recorded and the hardware is installed, a user will be required to identify his or herself to the system and present their finger for comparison. The complete fingerprint authentication process can be separated into five distinct steps: (1) obtain information, (2) fingerprint cleaning, (3) feature extraction, (4) fingerprint comparison, and (4) system response [5.6, 5.18].

The first step in fingerprint identification is obtaining input from a user. This input includes identification and a fingerprint sample. Identification often taken the form of personal access numbers or magnetic cards encoded with the owners identity. The authentication system uses the identification information to obtain a verification fingerprint for the comparison process. Before the comparison can begin, however, the new fingerprint sample must be processed.

The second step of the authentication process involves cleaning the scanned fingerprint so that the features of the print may be located. The cleaning process begins with the conversion of the print to a representation that the system can manipulate. The fingerprint is converted to a large matrix of 0's and 1's [5.18]. The 0's represent the valleys, or low points, of the print and the 1's represent the ridges, or high points, of the print. This results in a rough fingerprint pattern with ridge gaps and smudges; a result of dirty or damaged fingers. These irregularities are mathematically removed and the complete print is smoothed and thinned.

The third step, feature extraction, involves the location and demarcation of specific pattern features such as forks and ridge ends. Forks are the points of a fingerprint where two ridges join together and continue as one. Ridge ends are the points of a fingerprint where a ridge stops. [Figures 5.2a](#) and [b](#) illustrate examples of forks and ridges.



Figures 5.2 a & b. Illustrations of a fingerprint fork and ridge ends, respectively.

Once every fork and ridge end is located, their relative positions are recorded. Recording the points in terms of their relative positions to each other allows the authentication process to account for inconsistent alignments and orientations of finger placement on the pad.

In the fourth step, fingerprint comparison, uses the identification information and the fingerprint sample to determine whether the user is authorized to use the system. The system uses the identity information to select the fingerprint that will be the basis for access testing. Characteristics of the fingerprints are compared and the results used to determine whether the user should be granted access.

The final step of the process, system response, occurs when the authentication system either grants or denies a user access. In either case, the system, like the voice print system, logs the time and date access is attempted. This log assists in identifying the users who are regularly denied access and having difficulty with the system. Unlike voice print authentication logs, these logs are unnecessary for locating compromised accounts. This is a direct result of the difficulty in forging fingerprints. With rapid advancements in technology, however, fingerprint authentication may need to take additional biometric measurements, such as finger size or temperature, to decrease the possibility of unauthorized access.

While forgery is currently not a concern with fingerprint authentication, remote access to a system is a concern. Like magnetic cards, the only way a user will be able to authenticate themselves is with the use of additional hardware. Providing authorized users portable fingerprint readers is possible and becoming more practical with the recent advent of keyboards with built-in readers. The concerns, however, relate to the trustworthiness of any portable hardware. The fingerprint pattern produced by the hardware cannot be trusted as a knowledgeable user can alter the device to produce the

desired results. For these reasons, remote access is often denied or ensured by other authentication mechanisms.

5.2.3.3 Retinal Prints

Retinal authentication is the processes of determining system access based on the blood vessel pattern on the back of a user's eye [5.19]. Like fingerprint authentication, retinal scanning is exceptionally accurate and currently impossible to forge or duplicate. This makes retinal scanning an attractive authentication method.

Retinal authentication requires retinal scanners be placed at selected security points such as at each terminal or at the entrances to restricted labs. To obtain access to computing facilities protected with retinal scanners, each user must enroll in the authentication system and have their retinal pattern stored in the system. Because the process is very accurate and a user's retinal image does not change or age, only one scan is required to enroll.

Once the retinal print authentication system is in place, users will be required to provide identification information and look into a retinal scanner to gain access to the system. Like other biometric authentication methods, identification can take the form of either a personal access number or magnetic card. The authentication system uses this identification information to recall a user's retinal verification pattern. The verification pattern becomes the yardstick by which a user's retinal print will be measured. This pattern is obtained by scanning a user's eye with a low level infrared beam [5.5, 5.19]. The beam makes the blood vessels on the back of the eye appear in varying degrees of gray on a light background [5.5]. The light intensity at hundreds of predetermined points on the retina is recorded for comparison.

The authentication system compares the recorded intensities of the two retinal patterns to determine whether they agree with each other. When the patterns agree, the user is the person they claim to be and may be granted access. In the event that the two patterns do not agree, however, the user is not given access. In either case, the authentication system logs all attempted accesses.

Unlike other biometric access controls, the greatest problem with retinal scanning is not accuracy, rather user acceptance. As such, retinal scanning currently has few uses or installations. It is primarily found in limited quantities, controlling physical access to restricted areas. As societal views change, however, the use of retinal scanning is growing. The banking industry is testing retinal scanning, as well as fingerprint identification, in ATM banking terminals.

5.2.3.4 Hand Geometry

Hand geometry authentication relies upon hyper-accurate measurements of a persons' fingers and finger-webbing to determine whether a user should be granted access to a system, or more likely, a physically secured area [5.8, 5.19]. Unlike other biometric authentication devices, the physical size of many hand geometry measurement devices is too large to conveniently place at each terminal; most hand measurement devices measure approximately eight inches in each dimension.

The hand geometry authentication process begins with a user presenting an unverified identity in the form of either a personal access number or magnetic card. The system uses this identity to locate the expected hand measurements for the user. Along with the unverified identification, a user must place their hand on the measurement device. The user comfortably places their fingers between, and next to a series of pegs protruding from the device [5.24]. Each of the user's fingers must also lay across a guide on the device [5.19, 5.24]. An example hand placement is shown in [Figure 5.3](#), below. Once the hand is properly placed, sensors below the uncovered portion of the finger guides will record hyper-accurate measurements of the length of the user's fingers [5.13, 5.19].

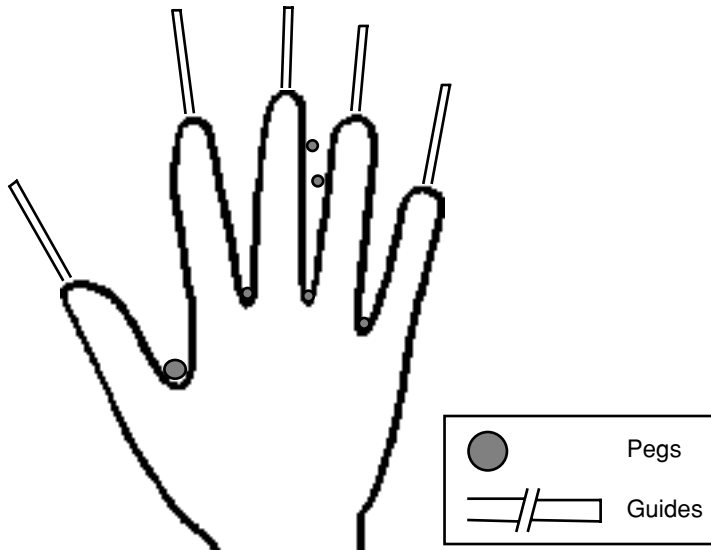


Figure 5.3. Proper hand placement on a hand geometry measurement device.

Once the hand measurements have been recorded by the system, they are compared to the expected values of the identity entered by the user. For the user to gain access to a system or restricted area, the measurements must fall within acceptable tolerances. As with other biometric authentication methods, keeping a log of the attempted accesses, whether successful or failed, will assist in locating problems and subversive behavior. Log files will provide assistance in locating and resolving problems that may occur with the hand geometry authentication system.

5.2.3.5 Signature Analysis

As signature analysis relies more on habitual actions rather than less-controllable physical attributes, some researches do not consider it a biometric measurement [5.7, 5.15]. A person's signature, however, relies on biometric characteristics as well as habitual actions [5.7, 5.12]. The manner in which a person writes their signature depends upon the size and development of their arm and muscles [5.12]. The pressure placed upon the writing instrument also depends on the development of a person's muscles [5.12]. Regardless of whether signature analysis is a biometric of many personal traits or a habitual action, it is widely used for user authentication and therefore discussed here.

Signature authentication examines the characteristics of how a user writes their signature. For the authentication system to grant a user access, the signature sample provided by the user must reasonably match in content as well as technique. These features are measured by a specially designed stylus (pen) and pad that must be installed at each terminal or access point.

Signature analysis authentication requires the user to provide numerous signature samples for comparison. These signature samples should be written in a normal fashion such that they can be easily duplicated by the original writer. When normally written, these signatures rarely need updating.

Once the system has been installed and the users have recorded samples of their signatures, they will begin using signature authentication to gain access to a computer system or restricted area. Like other authentication methods, the signature authentication process may be described as a series of steps: (1) obtaining information, (2) measuring characteristics, (3) comparing signatures, and (4) system response [5.12].

The first step of the authentication process is obtaining the necessary access information. To access a system, the user must provide an unverified identity and sample signature. Like the previous authentication methods, the unverified identity will most likely take the form of a personal access number or magnetic card.

The second step in the authentication process, measuring attributes of the signature, occurs both during and after the signature is written. Measured characteristics include the duration of the signature, pen pressure, pen acceleration, pen up-and-down

pattern, and the order in which pen strokes were made [5.12, 5.16]. After the signature is complete, line slopes, signature width, and signature content are also measured and recorded [5.12, 5.16].

The third step in the authorization process occurs when the signature measurements are compared to the authenticated signature measurements associated with the unverified user identity.

The final step in the process determines how the system responds to the user's request for access. For a user to be granted access, the signatures must be sufficiently similar by comparison. As with the other authentication methods, an access attempt log is kept by the authentication system. Access logs are especially important in a signature authentication system, because this access method is especially prone to attack.

The characteristics that will improve the authentication's ability to identify and reject forgeries are timing and pressure measurements [5.16]. Most forgeries are written slowly as the forger is attempting to create a visibly similar signature [5.16]. Sensitive timing constraints will reject this type of signature as being too slow. In the event that an unauthorized user is capable of producing a reasonable forgery within the necessary time constraint, the pressure placed upon the pen should identify the signature as a forgery [5.16]. Pressure measurements have proven to be a reliable means of identifying users as pen pressure is a difficult writing characteristic to reproduce accurately [5.16].

5.3 Summary

Most computer systems have a first line of defense; the authentication system. The purpose of the authentication system is to prevent unauthorized users from using the system and accessing the information within the system. This is accomplished by requiring all users to provide either secret or unique information. Common examples of secret information are passwords and questionnaires. Examples of unique information include smartcards, calculators, and biometrics such as voice prints and finger prints. Because the degree of security provided by the different authentication systems vary, system and security administrators will have to determine the most appropriate means of authentication for their environment. This will require a careful evaluation of both the needs of the computing environment and the advantages and disadvantages, of each system. Special consideration will have to be made for users requiring remote access to a system that employs some of the more advanced and robust systems.

5.4 Projects

- 5.1 Define and describe another means of authentication. Identify its strengths and weaknesses. Create a program which implements this authentication process.
- 5.2 Identify and detail a means of using many of the biometric authentication methods in remote situations. Are these methods practical enough to be implemented?

5.5 References

- 5.1 American National Standards Institute, *American National Standard ANSI/ISO 7810-1985: Identification cards – physical characteristics*, New York, New York, May 20, 1988.
- 5.2 American National Standards Institute, *American National Standard ANSI/ISO 78106/21988: Integrated circuit(s) cards with contacts – part 2: dimensions and location of contacts*, New York, New York, December 12, 1990
- 5.3 Brown, Michael K.; McGee, Maureen A. and Rabiner, Lawrence R., “Training Set Design for Connected Speech Recognition”, IEEE Transactions on Signal Processing, Vol. 39, No. 6, June 1991, pp. 1268-1281.
- 5.4 Campos, J. C.; Linney, A. D. and Moss, J. P., “The Analysis of Facial Profiles Using Scale Space Techniques”, Pattern Recognition, Vol. 26, No. 6, June 1993, pp. 819-824.
- 5.5 Chinnoek, Chris, “Eye-based systems show promise for security use”, Military and Aerospace Electronics, November 1994, pp. 6-11.
- 5.6 Chong, Michael M. S.; Gay, Robert K. L.; Tan, N. H. and Liu, J., “Automatic Representation of Fingerprints for Data Compression by B-Spline Functions”, Pattern Recognition, Vol. 25, No. 10, October 1992, pp. 1199-1210.

- 5.7 Davies, D. W. and Price, W. L., Security for Computer Networks, 2ed., John Wiley & Sons Publishing Co., New York, New York, 1984.
- 5.8 Fites, P. and Kratz, Martin P. J., Information Systems Security: A Practitioner's Reference, Van Nostrand Reinhold, New York, New York, 1993.
- 5.9 Garfinkel, Simson and Spafford, Gene , Practical UNIX Security, O'Reilly & Associates, June 1991.
- 5.10 Harmon, L. D.; Khan, M. K.; Lasch, Richard and Ramig, P. F., "Machine Identification of Human Faces", Pattern Recognition, Vol. 13, No. 2, February 1981, pp. 97-110.
- 5.11 Harmon, L. D.; Kuo, S. C.; Ramig, P. F. and Raudkivi, U., "Identification of Human Face Profiles by Computer", Pattern Recognition, Vol. 10, No. 3, March 1978, pp. 301-312.
- 5.12 Herbst, N. M. and Liu, C. N., "Automatic Signature Verification Based on Accelerometry", IBM Journal of Research and Development, Vol. 21, No. 3, May 1977, pp. 245-253.
- 5.13 Holmes, James P.; Wright, Larry J. and Maxwell, Russell L., *A Performance Evaluation of Biometric Identification Devices*, Sandia Nation Laboratories Technical Report SAND91-0276•UC-906, June 1991.
- 5.14 Kashyap, R. L., "Speaker Recognition from an Unknown Utterance and Speaker-Speech Interaction", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, No. 6, December 1976, pp. 481-488.
- 5.15 Morris, Robert and Thompson, Ken, "Password Security: A Case History", Communications of the ACM, Vol. 22, No. 11, November 1979, pp. 594-597.
- 5.16 Nagel, Roger N. and Rosenfeld, Azriel, "Computer Detection of Freehand Forgeries", *IEEE Transactions on Computers*, Vol. C-26, No. 9, September 1977, pp. 895-905.
- 5.17 Rao, C. V. Kameswara, "On Fingerprint Pattern Recognitions", Pattern Recognition, Vol. 10, No. 1, January 1978, pp. 15-18.

- 5.18 Rao, T. Ch. Malleswara, "Feature Extraction for Fingerprint Classification", *Pattern Recognition*, Vol. 8, No. 2, February 1976, pp. 181-192.
- 5.19 Russell, Deborah and Gangemi Sr., G. T., Computer Security Basics, O'Reilly & Associates, July 1991.
- 5.20 Samal, Ashok and Iyengar, Prasana A., "Automatic Recognition and Analysis of Human Faces and Facial Expressions: A Survey", *Pattern Recognition*, Vol. 25, No. 1, January 1992, pp. 65-77.
- 5.21 Svigals, Jerome, "Smartcards – A Security Assessment, Computers & Security", Elsevier Advanced Technology, Vol. 13, No. 2, April 1994, pp. 107-114.
- 5.22 White, George M. and Neel, Richard B., "Speech Recognition Experiments with Linear Prediction, Bandpass Filtering and Dynamic Programming", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. ASSP-24, No. 2, April 1976, pp. 183-189.
- 5.23 Wilpon, Jay G.; Rabiner, Lawrence R.; Lee, Chin-Lee and Goldman, E. R., "Automatic Recognition of Keywords in Unconstrained Speech Using Hidden Markov Models", *IEEE Transactions on Acoustics, Speech, and Signal Processing*, Vol. 38, No. 11, November 1990, pp. 1870-1990.
- 5.24 Wilson, Bill, "Hand geometry boasts simplicity, convenience", *Access Control*, March 1992.

5.6 Extended Bibliography

- 5.25 Boccignone, G.; Chianese, A.; Cordella, L. P. and Marcelli, A., "Recovering Dynamic Information from Static Handwriting", *Pattern Recognition*, Vol. 26, No. 3, March 1993, pp. 409-418.
- 5.26 Brault, Jean-Jules and Plamondon, Réjean, "A Complex Measure of Handwritten Curves: Modeling of Dynamic Signature Forgery", *IEEE Transactions on Systems, Man and Cybernetics*, Vol. 23, No. 2, March/April 1993, pp. 400-413.

- 5.27 Guerfali, Wacef and Plamondon, Réjean, "Normalizing and Restoring On-Line Handwriting", *Pattern Recognition*, Vol. 26, No. 3, March 1993, pp. 419-431.
- 5.28 Huang, C. L. and Chen, C. W., "Human Facial Feature Extraction for Face Interpretation and Recognition", *Pattern Recognition*, Vol. 25, No. 12, December 1992, pp. 1435-1444.
- 5.29 Ikeda, Katsuo; Yamamura, Takashi; Mitamura, Yasumasa; Fujiwara, Shiokazu; Tominaga, Yoshiharu and Kiyono, Takeshi, "On-Line Recognition of Hand-Written Characters Utilizing Positional and Stroke Vector Sequences", *Pattern Recognition*, Vol. 13, No. 3, March 1981, pp. 191-206.
- 5.30 Liu, C. N.; Herbst, N. M. and Anthony, N. J., "Automatic Signature Verification System Description and Field Test Results", *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-9, No. 1, January 1979, pp. 35-38.
- 5.31 Moayer, B. and Fu, K. S., "An Application of Stochastic Languages to Fingerprint Pattern Recognition", *Pattern Recognition*, Vol. 8, No. 2, February 1976, pp. 173-179.
- 5.32 Paquet, Thierry and Lecourtier, Yves, "Recognition of Handwritten Sentences Using a Restricted Lexicon", *Pattern Recognition*, Vol. 26, No. 3, March 1993, pp. 391-407.

6

ACCESS AND INFORMATION FLOW CONTROLS

In Chapter 5 we discussed identification and authentication methods used to limit access to a computer system. Early computers had no internal controls to limit access and any user could thus interact with any portion of the system desired. In today's larger and much more complex environments, however, just because individuals have been confirmed as authorized users does not mean that they should have access to all information contained on the system. It is the job of the *Access Control* security services to determine which information an individual can access.

Conceptually, what access controls are designed to do can be depicted as a protection table or matrix such as the one illustrated in [Figure 6.1](#). The table contains every user (or subject) who may have access to the system on one axis and every file (or object) stored on the system on the other axis. The entries located at the intersection of any user/file pair shows what type of access this user has for a specific file. Typical access modes which might be found in the table include:

- Read - allow the user to read the file or view the file attributes.
- Write - allow the user to write to the file, this may include creating, modifying, or appending onto the file.
- Execute - the user may load the file and execute it.
- Delete - the user may remove this file from the system.
- List - allows the user to view the file's attributes.

Often some of these may be implied in another. A user given *Read* access to a file will generally also be able to view that file's attributes. In [Figure 6.1](#) User2 has *Read* and *Write* access for File1 but only *Read* access for File2. We may also be able to assume that User2 can view both files attributes even though *List* is not specifically stated. The table lists User2's access to File3 as *List*, however, which means User2 can view the attributes of the file but not the file itself. A location in the table with no

entry means that the user has not been granted any type of access for this file. If desired, the table can be expanded to also include a list of other objects besides files such as specific hardware devices. This allows access to devices to be controlled. The last two columns of Figure 6.1 which includes the Printer and Disk, illustrates this concept. While a protection table may seem like a simple solution to the access control problem, in reality it is seldom implemented in this fashion. The reason for this is that the table created is extremely large and sparsely populated. Most users will have access to only their own files and not to the files created by another user. This results in a table in which most entries will be blank. Reserving space for a large, mostly empty table, is a tremendous waste of space. Other methods have consequently been created to address the problem.

	File1	File2	File3	File4	File5	Printer	Disk
User1			Read Write		Read Write	Write	
User2	Read Write	Read	List			Write	
User3		Read Write		Execute		Write	Read Write

Figure 6.1. A Protection Table [6.5, 6.6].

Working in conjunction with Access Controls are **Information Flow Controls**. These controls incorporate methods to regulate the dissemination of information among the subjects and objects on the system. The policies that dictate the permissible flow of information center around the security model chosen for the system (see Chapter 4 for a discussion of security models). While Access Controls play a major role in controlling the interaction between subjects and objects, they are not sufficient to regulate all aspects concerning the flow of information. Discretionary Access Controls, as defined in the DOD **Trusted Computer Systems Evaluation Criteria**, for example, will clearly delineate who may access a file but do not cover what the person does with the information once it is viewed. The user may make a copy of the accessed information and then pass it along to others not originally intended to receive it. The flow of this information may then exceed its intended scope. Mandatory Access Controls, on the other hand, also govern how the accessed files may be manipulated using operations such as copy, and thus restrict the flow of information. We will begin our discussion of Access and Information Flow

Controls with a discussion of various techniques to govern initial access to the objects protected by the computer system.

6.1 File Passwords

The use of passwords as an access control technique is similar to their usage for controlling access to the computer system itself. In order to gain access to a file, the user must present the system with the file's password. This password is separate from, and should be different than, the one used to gain access to the system. In this technique each file is assigned a password and each user who is to be given access to the file must be told the file's password. The original assignment of the password can be accomplished by the system manager or the owner of the file itself. In order to control the type of access granted to the file, multiple passwords for each file may be necessary. The system, for example, might use one password to control reading and a separate one to control writing.

While this technique is easy to understand and fairly easy to implement, it is cumbersome and suffers from several problems that renders it unsuitable for most computing environments. The first problem is with the passwords themselves. Since users have to remember a separate password for each file they have access to, this will result in a large number of passwords that have to be memorize. Most people will not, however, attempt to memorize the large number of passwords but will instead copy them and keep them on a piece of paper which they may tape to their terminal, keep in the top drawer of their desk, or store in their wallet. Obviously this defeats the purpose of having the passwords in the first place. In addition, this technique places a great amount of responsibility on the users to not share file passwords with individuals not authorized to them. Users are notorious for disregarding security policies when they are pressed for time and any technique that is so incumbent on the users themselves will usually result in numerous violations.

The second problem with password file protection is that there is no easy way to keep track of who has access to a file. The passwords are distributed manually, often by word of mouth, which leaves no automated trail to keep track of them. This lack of any kind of list describing who has access to a specific file leads to another problem, revoking a user's access to a file. Actually it is easy to revoke a user's access to a file, simply change the password. The problem is doing so without affecting all of the other users who need the file. Since there is no list of authorized users, when the file's password is changed, each of the other users will have to identify themselves to obtain the new password. This problem can be partially solved through the use of multiple passwords for each file. In essence, this creates groups of users. Each file may have a number of groups that have been given access to it. If a user's access is to be revoked,

only the group that the user is part of will be affected. All other groups may still access the file using their own password. A group password technique is especially useful when a user is to be granted access for a limited time. A group consisting of only the single user can be created and the password eliminated when that user's need for access to the file is over. There is obvious tremendous overhead associated with a multiple password file technique, which makes it cumbersome to use.

The final drawback to a password based file protection scheme is encountered when we have a file that requires access to other files (e.g., programs). One way to handle this is to embed the passwords for the other files inside the program itself but this results in a user who is granted access to the program being immediately granted access to the other files as well. Another way to handle this problem would be to require the user to specify all file passwords up front but this assumes that the user knows which files will need to be accessed before the program is executed. A third method is to have the user supply the password for each additional file as it is accessed but this method is cumbersome and slows processing. There really is no suitable way to handle the problems associated with file passwords.

6.2 Capabilities Based

A capabilities based access control scheme can be thought of as dividing our protection table from [Figure 6.1](#) by rows. Associated with each user is a list of the objects the user may access along with the specific access permissions for each object. This list is called a Capability List with each entry being a Capability. An example of the capability list for User2 in our previous example is shown in [Figure 6.2](#).

Object	Permissions
File1	R,W,--,L
File2	R,--,--,L
File3	--,--,--,L
Printer	--,W,--,--

Figure 6.2. Capability list for User 2 [6.6].

Instead of the actual name of the object, a capability list may instead store a pointer to the object. An additional field may then be added which would describe the type of object that is pointed to. The permissions field in [Figure 6.2](#) can be implemented using a bit map with each bit representing one of the possible access

modes or permissions. In our example, the possible access modes are R (read), W (write), E (execute), D (delete), and L (list). Notice that the L bit is set whenever the R bit is. This illustrates the point that a user will have List access whenever Read access is granted.

Capabilities can also be implemented as a *ticket* possessed by the subject which will grant a specified mode of access for a specific object. The system maintains a list of these tickets for each subject. This approach also allows a user to give access to an object to another user by passing copies of the ticket. Some systems may allow a user to revoke access to files they own by recalling the tickets given to other users.

Just like password file access techniques, both capability lists and capability tickets suffer from some cumbersome operational problems. The system is forced to maintain a list for each subject. A single object may be accessible to all or a large number of users/subjects and will thus have its access information or ticket repeated many times for each of these users/subjects. This leads to tremendous overhead as the system must maintain capability lists with large amounts of repeated information. In addition, systems employing capability techniques have no efficient way of knowing exactly which subjects have access to a specific object. Revoking access to a file is also a cumbersome proposition since the ticket to grant access may have been passed to a number of different individuals. The system will need to check every capability list to revoke access. A method which makes it easier to revoke access is to have the pointer in the capability list point to an indirect object which then points to the object itself. For an immediate revocation of access, the indirect object pointer can be deleted, thus breaking the chain. Of course this revokes everyone's access including those who may still legitimately require access.

To understand both the capabilities list and ticket techniques it may be useful to use an analogy. We can compare these techniques for granting file access to an attempt to secure a file cabinet by employing a guard to monitor it. Individuals who are to be granted access to a file within the cabinet will be given a special pass (the tickets) to show the guard. The ticket only grants access to a specific file and many users may have a copy of the ticket. The capability list, which contains a list of all files in the cabinet the user is authorized access to, is similar in that it will be shown to the guard who checks and upon verifying that the list includes the file cabinet, allows the user access. Obviously capability techniques need to include measures to keep users from forging tickets and lists so they can't obtain access to objects they are not entitled to. A simple method to do this is to make capabilities accessible only to the operating system and provide a very limited user capability to manipulate them.

6.3 Access Control Lists

While capability techniques can be thought of as dividing our protection table example into rows, Access Control Lists (ACLs) divide it by columns. Instead of maintaining a separate list for each subject detailing the objects that a subject has access to, ACLs are created for each object and list the subjects that have permissions to access them. There are several different ways that ACLs can be implemented. One example using our protection table from [Figure 6.1](#) is depicted in [Figure 6.3](#).

File1		File2		File3		File4	
User2	RWL	User2	RL	User1	RWL	User3	E
		User3	RWL	User2	L		

File5		Printer		Disk	
User1	RWL	User1	W	User3	W
		User2	W		
		User3	W		

Figure 6.3. Access Control Lists.

There are several obvious advantages in using Access Control Lists over the other techniques previously described. The first is the ease in which a list of all subjects granted access to a specific object can be determined. In a password scheme there was no way to accomplish this since passwords could be passed by word of mouth. In a capabilities-based scheme all of the subjects capability lists would have to be scanned to determine which had access to a specific object. Now the ACL is simply consulted. A second advantage is the ease with which access can be revoked. The owner of the object can simply remove (or change) any entry in the ACL to revoke or change the type of access granted to any subject. A third advantage over protection tables is the storage space that is saved using this method (although an even more storage conscious scheme will be discussed later). The tables do not have to list each subject for every object, just those that have access permission and the specific mode of access granted. One drawback to this method, traditionally not thought of as such because it is seldom requested, is the difficulty in listing all objects a specific subject has access to. In this case, all of the ACLs would need to be scanned to determine which contained a reference to the subject in question. A capabilities-based scheme, on

the other hand, could accomplish this very quickly by simply providing the capability list for the subject in question.

An approach often used on systems to aid in access control is to divide the users into groups which would coincide with real-world divisions such as all individuals working on a specific project. This additional feature can easily be accommodated with ACLs. Figure 6.4 depicts an ACL with groups added in a fashion similar to one used by the Multics and VMS operating systems [6.2]. In this example each entry consists of a user name and a group separated by a period. The user SMITH in group SALES has been granted *Read*, *Write*, and *List* access to File1. The '*' is used as a wildcard which matches any user or group so the second entry states that any user in group SALES has *Read* and *List* access to File1. Specific users can be denied access as depicted in the third line which states that user JONES in group PERSONNEL may not access this file at all. The last entry states that all other users are granted *List* access only.

File1	
SMITH.SALES	RWL
*.SALES	RL
JONES.PERSONNEL	N
.	L

Figure 6.4. An ACL employing Groups [6.2, 6.6].

One further feature which can be easily added to Access Control Lists is to provide the ability to restrict access based on factors such as the time of day or the location the user is accessing the system from. This can be done by either specifying when and where access is to be granted or providing times and locations when they are to be denied. The only real difference between the two is the default settings that result. Figure 6.5 shows how these can be added to specify any restrictions on time and location for the ACL of File1. In this example, User SMITH in group SALES has been given access to the file at any time from any location since no restriction is listed. User JOHNSON in group PERSONNEL can only access the file from 8:00 AM to 6:00 PM and can only do so from a local terminal. Should user JOHNSON attempt to access the file in the middle of the night, or from a dial-in line, access would be denied.

File1			
SMITH.SALES	RWL		
JOHNSON.PERSONNEL	R	0800-1800	Local

Figure 6.5. ACL with time and location restrictions [6.1].

Due to the fine granularity implemented in ACLs, the example in [Figure 6.5](#) could easily be enhanced to allow groups or users different modes of access based on the location and time of access. [Figure 6.6](#) shows a modification where user SMITH in group SALES is allowed *Read*, *Write*, and *List* access to File1 during normal business hours (8:00 AM to 6:00 PM) and can access it from any location during this time. After hours, however, SMITH only has been granted *Read* and *List* access and must do so from a local terminal.

File1			
SMITH.SALES	RWL	0800-1800	
SMITH.SALES	RL		Local
JOHNSON.PERSONNEL	R	0800-1800	Local

Figure 6.6. ACL with multiple individual user access restrictions [6.1].

6.4 Protection Bits

A modification to the Access Control List Approach uses only a few bits to describe the access permissions for each object. Similar to ACLs, Protection Bits are attached to each file but instead of providing a complete list of all users and their allowed access modes, they specify permissions for specific classes of users. [Figure 6.7](#) shows how Protection Bits can be used on a system to implement an *Owner/Group/World* scheme for our *Read/Write/Execute/Delete/List* access modes. Instead of a large list being attached to each object, the Protection Bits use only fifteen bits of memory. The fifteen bits are broken into three sets of five. The first set of five bits represents the permissions for the Owner of this file. The second set are used to delineate the access permissions given to users in the same group as the owner of the file. The final set of bits describe the permissions for all other users on the system. The first bit in each set of five is used to grant *Read* access mode permission. If the bit is set to one, *Read* access is granted. If the bit is set to zero,

Read access is denied. The second bit of each group is used for *Write* access, the third for *Execute*, and so forth.

OWNER					GROUP					WORLD				
R	W	E	D	L	R	W	E	D	L	R	W	E	D	L

Figure 6.7. Protection Bits for Owner/Group/World scheme.

The most common example of the use of Protection Bits is the UNIX operating system. UNIX systems employ an Owner/Group/World strategy for their file access with three access modes specified: *Read*, *Write*, and *Execute*. (Directories have a separate interpretation of what each of these access modes mean. Please see the projects at the end of this chapter.) Associated with each file on a UNIX system is the set of nine required access bits to implement this scheme. Each file also has an owner specified and a group which the user was part of when the file was created. A user is able to be part of many different groups and can freely change between these groups during a session by executing a command to change the current group. This enables the user to limit the access granted to any specific file to a designated number of individuals in a particular group. The result of this is to allow the user to actually limit access to a particular file to only a very few people by defining a special group for these select few to be part of. While extremely flexible, this method is somewhat cumbersome as users have to be aware of which group they are currently executing within. Often, because of the cumbersome nature of groups, the user simply gives everybody (the UNIX World category, sometimes on different systems referred to as Other or All-Other permissions) access to the file. The cumbersome nature of the Group structure is the only real drawback to this technique which is far outweighed by the greatly reduced storage requirements for the access control information. A minor drawback is the lack of an easy way to list all objects for which a specific subject can access.

6.5 Controls for Mandatory Access

This chapter has addressed a variety of techniques that can be used to provide Discretionary Access Control. As was discussed in Chapter 4, Mandatory Access Controls place additional restrictions on access by attaching a label to all subjects and objects indicating the clearance or security level classification. While the purpose of Discretionary controls was to allow users to limit access to the files they owned (i.e.,

access was granted at the ‘discretion’ of the user), the purpose of Mandatory controls is to limit access to files based on the security classifications of the files. Even if a user wants to grant file access to another user, the system would not permit it unless the new user’s clearance was sufficient to allow it.

Besides placing restrictions on users as subjects, Mandatory Access Controls also limit what processes or programs can do as subjects. A program will not be able to change the security restrictions imposed by the rules governing clearances and classifications for itself or other objects even if the same user owns them. A program will also be restricted from creating and using a shared file to pass information to another subject which would violate the security restrictions. It should be remembered that Mandatory controls do not replace Discretionary controls, they enhance them and serve as an additional restriction on the access of objects and flow of information. Only if both the Discretionary and Mandatory restrictions are followed will a subject be allowed access to an object.

An interesting problem which Mandatory Access Controls partially solve is the issue of Originator Controlled (ORCON) data [6.3]. ORCON data is information that may only be passed to another with the permission of the owner (originator) of the information. Normal Discretionary controls would allow the originator to specify another individual but would not prevent the second individual from copying the file and passing it along. Mandatory controls, on the other hand, would allow the owner to grant access to another user by creating a new category of information. Mandatory controls operate at two levels, the security level (which can be thought of as a vertical structure separating levels such as Secret and Top Secret) and categories (which is a horizontal separation within a level containing such identifiers as NATO or Nuclear). Since only the originator and those other users the originator grants access to are authorized the special category, the Mandatory controls would prevent anybody from making copies of the data and passing it along. Organizations that handle a large volume of ORCON data, however, may find that the number of additional categories created to handle every instance of ORCON data can quickly exceed the capacity of the system. There are organizations in the Federal Government that handle thousands of different ORCON documents [6.3]. Mandatory controls were not designed to handle such a large volume of categories.

As a final comment, we should remember that while Mandatory Access Controls inevitably apply more severe restrictions on object access, this is not done to limit the user but rather to protect the objects. A possible threat to the system (addressed in Chapter 3) is covert channels. They may be used to transfer information to subjects not authorized to access it. The techniques discussed to address this threat (required in the Orange Book Mandatory Access Division) should be implemented to lessen the threat to Information Flow Controls. It is because the restrictions are so strong that Mandatory Access Controls actually progress a long way towards eliminating another problem that Discretionary Access Controls don’t effectively address; Trojan horses.

6.6 Trojan Horses

The four Access Control techniques previously described all have a major drawback when used to perform as Discretionary Access Controls or their equivalent. Programs that are executed by a user are free to access and modify all objects accessible by that user. This generally includes the ability to modify the access modes of the files. The operating system can't tell the difference between a legitimate request to change the permissions for an object owned by the user, and a request made by some malicious software to do the same. In this case, the specific malicious software we are concerned with are Trojan Horses.

The simple definition of a Trojan horse is software that appears to the user to be performing one function while it hides some other, often malicious, function. A more extensive definition is provided in Chapter 15, *Malicious Software*.

Limiting the ability of a Trojan Horse to perform its function requires several defensive measures. As was already mentioned, if the access control scheme employed by the computer system allows programs to modify the permissions of files the user owns then the Trojan Horse will be able to easily function on the system. Simply changing to more restrictive access controls which would cancel the ability of programs to modify permissions would not by itself be sufficient to eliminate the threat from Trojan Horses. Instead of modifying the access permissions, the Trojan Horse would now make copies of files in the Trojan Horse creator's own directory. If the ability to do this was also eliminated, the Trojan Horse might then send files via electronic mail to its creator so this ability needs to be curtailed as well. Of course by limiting the ability of the Trojan Horse to be able to function by restricting the possible ways that information may flow within the system, we are also severely restricting what legitimate users can do.

Another approach to address the Trojan Horse threat is to eliminate one of the requirements for a Trojan Horse. Eliminating the ability of users to develop programs would, for example, make it hard for them to create the Trojan Horse in the first place. This, however, severely limits the usefulness of the system and delegates it to operating only a select group of application programs. Placing voluntary procedural controls on the users to keep them from executing untested software is highly unreliable and doesn't address the possibility of Trojan Horses in system software. A particularly insidious Trojan Horse, described by McDermott [6.4], could be installed in the systems compiler itself. A Trojan Horse installed here would infect all other programs that are compiled and result in numerous other Trojan Horses. In this case, the compiler Trojan Horse does not perform the ultimate function but instead plants this function in the other programs that it compiles. The best way to eliminate this threat is to severely restrict access to the source code for the compiler and also to restrict the ability of individuals to compile and replace existing system software. This

particular Trojan Horse serves to illustrate the tremendous dependency we must place on the system software. Some organizations requiring extreme security measures go so far as to not accept any compiled vendor software. Instead they will only accept the source code and will compile the executable modules themselves. This, of course, means that they either have to trust the source code to not contain a Trojan Horse, or they must inspect the source code itself to insure that it does not contain one. Inspection of source code is a tedious process and it is extremely difficult to spot malicious code.

Perhaps the best way to limit the possibility of Trojan Horses is to implement the type of Access and Information Flow Controls used for Mandatory Access. While this will limit the ability of a Trojan Horse to send files or grant permissions to subjects at other security levels, it does not eliminate the problems associated with same-level copying of files. Assigning numerous categories to subjects and objects will further restrict the ability of the Trojan Horse to function but it also creates an environment which is more restrictive for the users. In the end, the best way to handle the possibility of Trojan Horses is to select the security model, enforce the resulting Access and Information Flow Controls, and to continually monitor the system to check for possible Trojan Horses.

6.7 Summary

Once authentication techniques have been used to validate a user's right to gain access to a computer system, it is up to the Access and Information Flow Controls to insure that the proper handling of information takes place. Access Controls determine what permissions a subject has for a particular object. Information Flow Controls handle what may happen to information once it has been accessed.

Several techniques were discussed to implement Access Controls. Perhaps the easiest is a File Password scheme in which access to any object is governed by a password for that object. The drawbacks to this method are that it is cumbersome (a user will have to remember many different passwords), there is no way to determine who has access to any given object since passwords can be transmitted by word-of-mouth, and there is no easy way to revoke a single user's permission to access an object without affecting other users as well.

The second technique described was a Capabilities-based scheme. In this technique, a list of Capabilities is kept for each user which show which objects the user has permission to access. The problems associated with this technique include the lack of a convenient method to determine who has access to a particular object without reviewing all user's capability lists. Depending on how the system is implemented, revoking a users access permissions may also be a difficult matter. There is also the

issue of wasted storage space associated with the use of capability lists since access information is duplicated numerous times among the various user's lists.

A third technique described was the Access Control List method which associated a list with each file instead of with each user. This makes it much easier to determine who has access to a specific object and also makes it extremely easy to revoke or change a single user's (or group's) access permissions. A drawback, not viewed as severe since it is not often requested, is the lack of an easy way to determine which objects a particular user has access to.

Perhaps the most common method, since it is employed in widely used operating systems such as UNIX, is the Protection Bits method. In this technique, a series of bits grouped in sets are associated with each object which describe the type of access the owner, individuals in the same group, and all other users have. Since only a few bits are associated with each file this technique requires very little overhead. It also is easy to revoke or modify access permissions and it is easy to determine what users have access to any particular object.

All of these techniques can be used to implement Access Controls and rudimentary Information Flow controls needed to implement protections at a level akin to Discretionary Access. In order to further prevent the unauthorized flow of information, further controls such as seen in Mandatory Access Controls are required. Controls at this level will keep programs from transferring information between security levels. These additional controls also help to address the threat of Trojan Horses since this type of malicious code will not be able to easily transmit information or change access permissions.

6.8 Projects

- 6.1 How Groups could be used to partition users in order to match real-world organizational structures was shown for both Access Control Lists and Protection Bit schemes of access control. How can groups be implemented in File Password and Capability based schemes? How could time-of-day and location restrictions be added?
- 6.2 In an Owner/Group/Other protection bit scheme, the first set of bits represent the permissions that owners may grant to themselves. Why might owners want to limit the permissions they give themselves?
- 6.3 How might the *Read*, *Write*, and *Execute* modes of access be interpreted for directory structures? If you have access to a UNIX-based system, determine whether your interpretations were correct.

6.9 References

- 6.1 Ford, Warick, Computer Communications Security: Principles, Standard Protocols and Techniques, Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- 6.2 Gasser, Morrie, Building A Secure Computer System, Van Nostrand Reinhold, New York, New York, 1988.
- 6.3 Graubart, Richard, “On the Need for a Third Form of Access Control”, *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 296-304.
- 6.4 McDermott, John, “A Technique for Removing an Important Class of Trojan Horses from High Order Languages”, *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988, pp. 114-117.
- 6.5 Silberschatz, Abraham and Galvin, Peter, Operating System Concepts, 4ed, Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.
- 6.6 Tanenbaum, Andrew S., Modern Operating Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1992.

6.10 Extended Bibliography

- 6.7 Abrams, Marshall; Heaney, Jody; King, Osborne; LaPadula, Leonard; Lazear, Manette and Olson, Ingrid, “Generalized Framework for Access Control: Towards Prototyping the ORGCON Policy”, *Proceedings of the 14th National Computer Security Conference*, Washington D.C., October 1991, pp. 246-256.
- 6.8 Boebert, W. and Ferguson, C., “A Partial Solution to the Discretionary Trojan Horse Problem”, *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, Maryland, October 1985, pp. 141-144.

- 6.9 Bonyun, David, "Rules as the Basis of Access Control in Database Management Systems", *Proceedings of the 7th National Security Conference*, Gaithersburg, Maryland, September 1984, pp. 38-47.
- 6.10 Denning, D.E. and Denning, P.J., "Certification of Programs for Secure Information Flow", *Communications of the ACM*, Vol. 19, No. 8, August 1976, pp. 461-471.
- 6.11 Foley, Simon N., "A Universal Theory of Information Flow", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 116-122.
- 6.12 Guttman, Joshua, "Information Flow and Invariance", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 67-73.
- 6.13 Karger, Paul, "Limiting the Damage Potential of Discretionary Trojan Horses", *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, April 1987, pp. 32-37.

AUDITING AND INTRUSION DETECTION

Audit trails were originally designed to be used for accounting purposes, not as an aid to security. Dorothy Denning presented a paper at the IEEE Conference on Security and Privacy in 1986, however, in which she described how audit trail data could be used to enhance the security of a computer system [7.2, 7.3]. Since that time, several research projects have resulted in programs which use a system's audit trail to help detect intrusive activities. These systems, known as Intrusion Detection Systems (IDS), have been extended to work in mostly limited networked environments. This chapter will examine this relatively new field in computer security and will examine several current systems to see how intrusive activity can be detected.

7.1 Audit Trail Features

In terms of system security, there are five goals for audit trails. These goals, as outlined in the Rainbow Series *A Guide To Understanding AUDIT in Trusted Systems* [7.10] are:

- To allow the review of patterns of access to individual objects, access histories of specific processes and individuals, and the use of the various protection mechanisms supported by the system and their effectiveness.
- To allow discovery of both user's and outsider's repeated attempts to bypass the protection mechanisms.
- To allow discovery of any use of privileges that may occur when a user assumes a functionality with privileges greater than his or her own.

- To act as a deterrent against perpetrators' habitual attempts to bypass the system protection mechanisms.
- To supply an additional form of user assurance that attempts to bypass the protection mechanisms are recorded and discovered.

Thus, audit data serves two main purposes; to detect unauthorized and intrusive behavior and to deter the same activities by its mere presence. There are two parts to auditable data at the C2 level of security: *auditable events* and *auditable information*. Auditable events are those actions which may be of concern from a security standpoint. Auditable information is the actual data relating to an occurrence of one of the specified auditable events. Examples of auditable events include:

- Use of identification and authentication mechanisms.
- Introduction of objects into a user's address space.
- Deletion of objects from a user's address space.
- Actions taken by computer operators, system administrators or system security administrators.
- All security-relevant events as outlined by the goals stated above.
- Production of printed output [7.10].

The type of information which should be collected on the occurrence of each of these auditable events include:

- Date and time of the event.
- The unique identifier for the subject generating the event.
- Type of event.
- Success or failure of the event.
- Origin of the request for identification/authentication events.
- Name of the object introduced, accessed, or deleted from a user's address space.
- Description of modifications made by administrators to a security database [7.10].

Thus, for example, all failed login attempts would be recorded with the date and time the attempt occurred. If an unauthorized individual does somehow gain access to the system, the audit trail would provide a record of the activities taken by the intruder, including the date and time of each action. While this would not restore files that are destroyed, it does help to determine the extent of the damage caused by the intruder. In addition, authorized users may be deterred from attempting to perform certain

unauthorized activities because they know that a record of their actions is being created. In this sense the audit trail feature serves to deter unauthorized activities.

A number of computer vendors have attempted to comply with the DoD C2 audit requirements. Sun Microsystems, for example, has released a version of their SunOS which includes optional facilities providing equivalent data to that specified in the AUDIT guide [7.4]. Sun uses a number of categories to specify the events the system administrator might want to audit. These categories are listed in [Table 7.1](#).

Table 7.1. Sun Microsystems categories of auditable events [7.4].

- | |
|---|
| <ul style="list-style-type: none">• dr - Reading of data, open for reading, etc.• dw - Write or modification of data.• de - Creation or deletion of an object.• lo - Login and logout.• ad - Normal administrative operation.• p0 - Privileged operation.• p1 - Unusual privileged operation. |
|---|

For each of the categories in [Table 7.1](#), the administrator can choose to log either a successful or unsuccessful attempt – or both. In addition, the security features allow selected users to be monitored without wasting disk space with data from other individuals who are not believed to be a threat. Thus administrators can closely monitor the actions of individuals whose current or past actions are suspicious.

The type of information that may be recorded by audit trails listed in [Table 7.1](#) is indicative of what is available from modern operating systems. The question remains, however, as to how this information is used. One can imagine the voluminous amount of information recorded for even a moderate sized system if all user accounts and activities are monitored. How does one go about searching through this data to find the few items that may indicate unauthorized activity? It is this problem that Dr. Denning addressed and which is the purpose of current Intrusion Detection Systems.

7.2 Intrusion Detection Systems

Intrusion Detection Systems can be designed to detect both attempted break-ins by outsiders as well as unauthorized activity performed by insiders. The types of activities these systems check, include:

- Attempted/successful break-ins.
- Masquerading.
- Penetration by legitimate users.
- Leakage by legitimate users.
- Inference by legitimate users.
- Trojan horses.
- Viruses.
- Denial-of-Service [7.2, 7.3].

Break-ins are one of the most widely known threats to computer systems. It is common to hear reports in the news regarding a computer system or network that has been penetrated by an individual or group of 'hackers'. A break-in is nothing more than an individual who is not authorized to use the system or network gaining access to it. Simply looking through an audit file for unsuccessful attempts to log on to a computer system is not enough since normal users will periodically make a mistake when they enter their password or userid, or they may occasionally forget them. Providing a list of unsuccessful login attempts is only the first step. Intrusion Detection Systems are designed to do more.

Masquerading often goes hand-in-hand with break-ins. It involves an individual, who has gained access to another's account, trying to appear as that individual. It may also involve such things as the modification of messages or files so that they appear to have been sent or modified by somebody else.

Also similar to break-ins are penetrations by legitimate users. This involves an authorized user of the computer system or network attempting to gain privileges or to perform actions which the user is not authorized to have or perform. A common example is a user attempting to gain system administrator privileges (such as logging in as *superuser* on a UNIX system).

Leakage of information by legitimate users is of special concern in systems which operate in a multi-level security mode. A user authorized access to Top Secret information who attempts to send a file to a user authorized access to only Secret information should be detected by an Intrusion Detection System. This may also involve other actions such as an individual attempting to print a file on a printer located in an area with individuals not authorized access to the file or a user attempting to store a file on a storage device not authorized to handle it.

Inference by legitimate users is an interesting issue. It involves the attempt by a user to obtain information the user is not authorized through the aggregation of several pieces of data. This is a common problem in multi-level databases where an individual may not be authorized access to all information stored in the database but can infer certain information from other entries in the database.

A Trojan horse is a program which appears to be designed for one purpose but actually performs another (often malicious) action. It in fact may also perform the actions it appeared to be designed for in the first place. An Intrusion Detection System often attempts to discover the hidden activity. One way that this can be done is by noticing the amount and type of resources the program used and comparing them with the amount and type that should have been used for the advertised purpose of the program. Also, ordinary sounding processes attempting actions in violation of established security policies might indicate something hidden inside the code.

Somewhat in a manner similar to Trojan horses, viruses are a threat that Intrusion Detection Systems search for. A virus is a program which attaches itself to other programs in order to replicate. It also may hide some malicious intent beyond its own propagation. Unusual file accesses by programs could be an indicator that a virus may have infected the system.

A final activity an IDS looks for is a denial-of-service attack. In this threat, the attacker attempts to monopolize specific resources in order to deny the use of them, and the system in general, to legitimate users. Unusual patterns of resource activities and requests may indicate such an attack is occurring. This is especially true if these requests have been generated from a user who is not normally associated with these specific requests. Denial-of-service attacks may, in some circumstances, originate from outside of the system or network. In these cases, the attack would still be indicated by an unusually high pattern of requests for specific actions (e.g., multiple logins) or network traffic (e.g., flooding a system with electronic mail messages).

Intrusion Detection Systems can use one of several methods to perform their assigned detection activities. In the past, they have been based on user or group profiles designed to look for specific user actions or attempts to exploit known security holes. Another method uses an intruder's profile as its basis of comparison.

7.2.1 User Profiling

The basic premise behind user profiling is that the identity of any specific user can be described by a profile of commonly performed actions. The user's pattern of behavior is observed and established over a period of time. Each user tends to use certain commands more than others, access the same files, login at certain established times and at a specific frequency, execute the same programs and so forth. A characteristic profile can be established for each user based on these activities and is

maintained through frequent updating. An intruder masquerading as an authorized user will generally not perform the same operations or be logged on at the same time as the authorized user. The intruder can thus be detected by comparing current activity against the previously established pattern of behavior exhibited by the user. Since the intruder's activity will fall outside of the established profile, the intrusion can be detected. Authorized users performing unauthorized activity can similarly be detected since the unauthorized activity would also fall outside of their normal profile. A variation of this method is to establish group profiles which describe the normal activities a member of a group would be expected to perform. Groups might be established, for example, along functional lines in an organization such as programmers, administrators, and various types of application program users.

Generic profiles are often assigned when a user is first given access to the system. This profile is then updated as the user interacts with the system for normal activities. One problem with profiling, especially when designed to detect unauthorized activity by authorized users, is the ability of the users to gradually change their profiles over time. Gradually extending the hours when a user normally logs on (over a period of several months) would allow the user to start logging in at midnight when the normal activity should have been during standard daylight hours. If the user had suddenly logged in at midnight, the activity would have been detected. By gradually extending work hours or the login time, the profile would also correspondingly change so the activity would not be recognized as abnormal. Outsiders can also escape detection in user-profile based systems as long as they don't perform too many activities or attempt to access unusual files.

7.2.2 Intruder Profiling

The idea behind intruder profiling is similar to law enforcement descriptions of profiles for certain types of criminals. Outsiders and unauthorized users will perform certain activities or act in a certain way when they gain access to a computer system. These activities, if they can be identified and put in the form of a profile, can be searched for and, if found, may indicate that intrusive activity is occurring. An example of this activity, frequently observed, occurs when an intruder gains access to a computer system for the first time. Intruders will often immediately check to see who else is logged on. They may then examine the file systems and wander through the directory structure, occasionally looking at files. They usually don't stay connected very long; simply logging on, performing a few actions, and then logging off. An authorized user normally doesn't act in such a seemingly paranoid manner.

7.2.3 Signature Analysis

Just as an individual has a unique written signature which can be verified through handwriting analysis, individuals likewise have a "typing signature" which can also be verified through keystroke analysis [7.5]. The time it takes to type certain pairs or triplets of letters can be measured and the collection of these *digraphs* and *trigraphs* together form a unique collection which can be used to characterize individuals. Certain digraphs and trigraphs, because of their frequency, can be considered more important and a minimal set of these important di- and trigraphs can be used in the identification process. In fact, a set of the five digraphs *in*, *io*, *no*, *on*, and *ul* have been found to be sufficient to distinguish touch typists from each other, with a reasonable level (95 percent confidence level) of accuracy [7.5].

Of course this type of identification technique requires specialized monitoring equipment which would be difficult to implement across a large network. Instead, however, another type of signature might be used, one in which common mistakes may be identified and used to help identify the authenticity of a user. While this technique is much less reliable than the keystroke analysis, it could be used to afford an additional level of security.

7.2.4 Action Based

Another approach to identify intrusive activity is "action based intrusion detection. In this approach, specific activities or actions which are known to be common activities performed by intruders are checked. For example, a common occurrence on UNIX systems is for an intruder to attempt to gain access to *root*, therefor an action based system might look for all occurrences of individuals attempting to become *root*. This does not mean that all individuals who become *root* are immediately tagged as being an intruder but rather limits the number of users whose actions must be fully monitored.

A similar method is to check for attempts to exploit known security holes. There are a number of common holes in systems and application software that can be exploited by knowledgeable individuals. An individual attempting to exploit any of these holes is therefor a suspected intruder. The obvious problem with this approach is that it requires knowledge of the most significant holes, or intruders will simply exploit those not monitored. The way to implement an action based system is to allow for easy addition of newly discovered holes.

Another type of action that might indicate intrusive activity is the attempted use of commands for a different operating system. New users to a computer system, unfamiliar with the specific operating system in use, may attempt commands they are used to performing on another system. Using VMS commands in a UNIX

environment, for example, might indicate an intruder has somehow gained access to the system. New users may make similar mistakes but established users should make fewer.

7.2.5 IDES

Originally developed in 1985 at SRI International [7.8], the Intrusion Detection Expert System uses a user profile approach to intrusion detection as well as an expert system to check for activities that match known attack scenarios or attempts to exploit known system vulnerabilities [7.8, 7.9]. The structure of IDES is shown in [Figure 7.1](#).

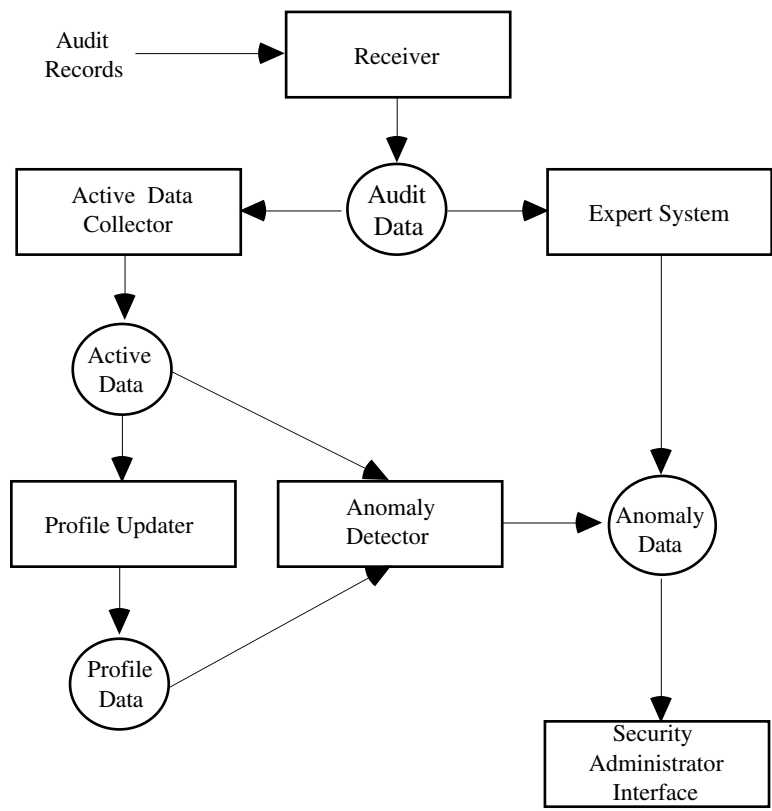


Figure 7.1. IDES Structure [7.8].

IDES compares the current behavior of a user to an historical profile to determine whether the current activity is within norms. If the current behavior is abnormal, it may indicate intrusive activity. IDES does not keep a record of all activity for each user but uses a subset of measures to describe normal activities. The measures chosen by the IDES researchers include [7.8]:

- CPU usage
- Input/Output usage
- Connect time
- Location of use
- Command usage
- Mailer usage
- Compiler usage
- Editor usage
- Directories accessed and/or modified
- Errors
- Hour and Day of use
- Network Activity

Some of the measures record a count of a specific activity while others simply record whether the activity occurred at all. An example of this is *command usage*. One measure records the number of times each command was used during a session, while another records (in a binary format) simply whether a command was used. The user profiles are updated once each day using a series of weights to give the most current data more influence over the new profile.

It is the arrival of the audit records that drives IDES. After receiving an audit record, the *Receiver* parses and validates it and places valid records into the *Audit Data*. The *Audit Data* database is used by both the statistical anomaly detector portion of IDES as well as the *Expert System*. In the statistical anomaly detector, the record is first used by the *Active Data Collector* to generate the *Active Data*. *Active Data* consists of information on all user, group, and remote host activities since the last time the profiles were updated. This data is in turn used by the *Anomaly Detector* to see if it indicates activity which could be considered outside of the norm when compared to the established user's individual or group *Profile Data*. If it is abnormal, an anomaly record is created and is stored in a database of *Anomaly Data* which can be accessed through the *Security Administrator Interface*. The level of sensitivity for the anomaly detector is set by the security administrator through this interface. Profile updating takes place daily and is performed by the *Profile Updater*.

Operating in parallel with the statistical anomaly detector is the *Expert System*. This system receives the *Audit Data* as did the *Active Data Collector*. An expert system analysis is also included in recognition of the fact that, while some users have a profile based on well established patterns of behavior, others have patterns which are sporadic and thus much harder to describe. With this sort of 'loose' profile almost any type of behavior would be viewed as normal and an intruder gaining access to such an account would probably go unnoticed. The *Expert System* looks for specific actions which can be viewed as indicators of possible intrusive activity independent of whether the action was considered outside of the users profile. An example of this type of action would be the execution of a command to query the status of a specific account followed by an attempt to log into that same account (e.g., a *finger* command followed by a *login* command) [7.8]. Another example would be a user attempting to exploit a known hole or vulnerability in the security of the system. The obvious weakness with this expert system approach is that only known attack methods and vulnerabilities are checked. Exploitation of unknown (to IDES) holes would go undetected.

An enhancement to IDES has extended its intrusion detection functions to a networked environment. Several interconnected hosts transfer their audit trail information to a central site which then uses the same methodology to determine whether intrusive activity is or has been taking place.

7.2.6 MIDAS

The Multics Intrusion Detection and Alerting System (MIDAS) is an intrusion detection system specifically designed to work with the Multics operating system [7.11] as used by Dockmaster. Dockmaster, NSA's unclassified computer system, is used to provide information on security related topics to the computer security community. MIDAS consists of several components as depicted in [Figure 7.2](#). A portion of the processing is performed on the Multics system while the rest is run on a separate Symbolics Lisp machine [7.11].

As an *audit record* is generated on the Multics system, the *Preprocessor* filters out data not used by MIDAS and formats the remaining information into an assertion for the fact base. This assertion is sent to the *Fact Base* via the *Network Interface* which links the Multics and Symbolics machines. The *Command Monitor* captures command data that is not audited by Multics and sends it to the *preprocessor* where it is handled in the same manner as the audit records. The introduction of an assertion into the *Fact Base* may cause a binding between this new fact and a rule contained in the *Rule Base*. It may, in fact, cause a series of rules to be activated. This assertion may thus change the state of the system and may result in a system response to a suspected intruder. The *Statistical Database* contains both

user and system statistics and defines what normal activity is for Dockmaster. Should the current record fall outside of the defined norm, a rule would be instantiated which in turn may change the state of the system and cause a system response.

It is obvious that how well MIDAS performs, is a function of the strength of the rules in the rule base. There are three types of rules used by MIDAS: *immediate attack heuristics*, *user anomaly heuristics*, and *system state heuristics* [7.11]. The *immediate attack heuristics* do not use any of the stored statistical data. They are designed to perform a superficial examination of the records, looking for events that by themselves indicate that a security violation may have occurred. *User Anomaly heuristics* use the statistical profiles to determine if the current activities of the user are outside of the norm defined by previous sessions. The *system state heuristics* are similar to the user anomaly heuristics except these are applied to the system itself and not a specific user. An example of this is an inordinate number of unsuccessful logins system-wide which might indicate an attempt by an individual to break into the system but which might not be noticed at the individual user level.

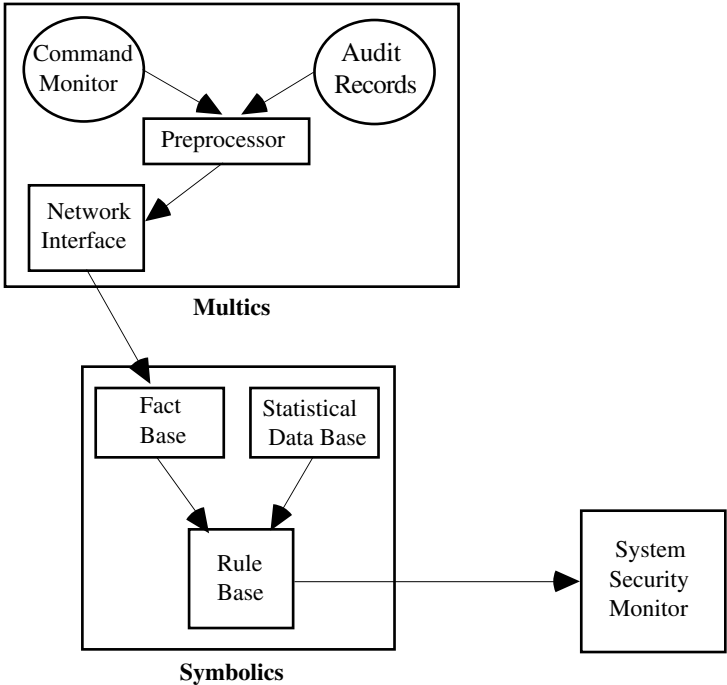


Figure 7.2. MIDAS Architecture [7.11].

7.2.7 Haystack

Haystack was developed for the Air Force as an audit trail reduction and intrusion detection tool [7.12]. Unlike the two previous systems, Haystack was not designed to work in a real-time environment but is run in an off-line batch mode. Haystack initially existed as two components, one on the Unisys 1100/60 mainframe and the other on a Zenith Z-248 IBM PC compatible machine (shown in Figure 7.3).

The audit trail on the Unisys 1100 first passes through a *Preprocessor* which extracts the necessary information required by the analysis program. This information is placed into a Canonical Audit Trail (CAT) file and is written to a 9 track tape. The tape is later read by a PC and the records processed.

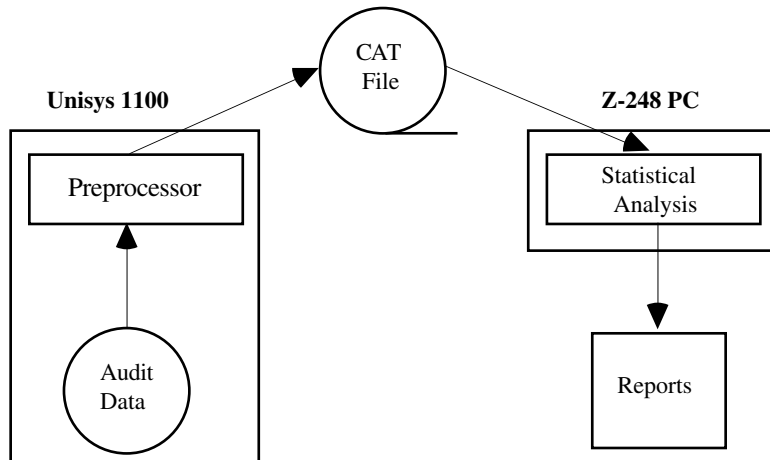


Figure 7.3. Haystack Organization [7.12].

The processing of the data is performed in two parts. Each record is checked to see if it indicates a security-relevant event and is logged if it does. The records are also used to update a database which contains information on the users past behavior. There are two types of statistical analysis that Haystack performs. The first is a trend analysis which uses the information in the database to determine if the current user's actions are significantly different from that of the past sessions. The other analysis compares the actions of a user's current session with a 'target intrusion', which is a description of the activities for one of the types of intrusions, Haystack is trying to

detect. This analysis will yield a 'suspicion quotient' which indicates how closely this session resembles a target intrusion.

Calculation of the suspicion quotient is a four step process. The first step entails the generation of a session vector \mathbf{X} from the audit records which represents important details about the current user's session [7.9]. These details consist of current session counts for specific attributes. Each element of this session vector is then compared with its corresponding threshold value found in a threshold vector. The threshold vector describes what the system considers normal activity. The result of this comparison is a binary vector indicating the session attributes which fall outside a 90 percent threshold range based on historical data. The binary vector \mathbf{B} is thus calculated as follows [7.9]:

$$b_i = \begin{cases} 0 & t_{i,\min} \leq x_i \leq t_{i,\max} \\ 1 & \text{otherwise} \end{cases} \quad (7.1)$$

The third step of the process consists of calculating a weighted intrusion score by summing the result of multiplying all elements in the binary vector with a corresponding value found in a weighted intrusion vector. Each element w_i of the weighted intrusion vector represents the relative importance of that attribute to detecting the specific target intrusion. The weighted intrusion score is then used to obtain the suspicion quotient by determining what percentage of all sessions have a weighted intrusion score less than or equal to the one just obtained for the current session.

An obvious drawback to the way Haystack performs intrusion detection is that it is not designed to be used in a real-time environment. This results in a considerable amount of time passing before an intrusion is noticed. A tremendous amount of damage could have occurred between the time an intruder gained access and the time the intrusion is detected. Another problem experienced by the developers of Haystack, a problem common to all profile based systems, was the determination of what attributes were significant indicators of intrusive activity. Despite these drawbacks, the Haystack algorithms used to determine whether an intrusion occurred have been used in other intrusion detection systems.

7.3 Network Intrusion Detection

The intrusion detection systems described up to this point were designed to detect intrusions on a single host. With the increasing number of systems that are part of a networked environment, it has become necessary to consider intrusions not only of single hosts but of networks as well. Some early attempts at solving this problem consisted of nothing more than taking existing individual host intrusion detection

systems and expanding their scope to a network. This was the case with both IDES and another system called the Information Security Officer's Assistant (ISOA). Other efforts have built upon the lessons learned in the single host intrusion detection systems while taking into account the additional characteristics found in a networked environment.

7.3.1 Network Attack Characteristics

Connecting a computer to a network increases the possibility that a system will be attacked since a larger number of users have access to it. At the same time, if the systems can work together, certain attacks will be easier to spot from a network perspective. For example, consider what is known as a 'doorknob rattling' attack. Similar to an individual who walks around a building checking each door in an attempt to find a door that was left unlocked, this attack consists of an intruder attempting to gain access to a computer system by repeatedly trying single logons. If unsuccessful, the intruder doesn't attempt to access this same system but moves on to another. In terms of the individual hosts, a single unsuccessful attack is not enough to raise suspicions. Observed from a network perspective, however, multiple hosts – each experiencing a single unsuccessful logon – would indicate a systematic network attack. Another aspect of the network environment that makes it significantly different than single host systems is that not only are there multiple hosts to worry about, but the hosts may not be of the same type nor will they all be individually monitored. By taking advantage of the broadcast nature of a network, unmonitored individual hosts can still be considered monitored (to a certain degree) if all traffic to and from them is monitored. In addition, all traffic on a network can be monitored without making significant changes to the operating system. Often this will provide more information than what is available on individual host audit trails.

7.3.2 NSM

The Network Security Monitor (NSM), developed at the University of California, Davis, does not use audit trails to perform its intrusion detection functions. Instead, it monitors the broadcast channel to observe all network traffic. As a result of this, it can monitor a variety of hosts and operating system platforms. A significant advantage to this method is that an intruder will not be able to tell that the network is monitored. This is an important point, since intruders have been known to turn off the audit trail features on systems that were known to be monitored [7.9]. The logical architecture of NSM is shown in [Figure 7.4](#).

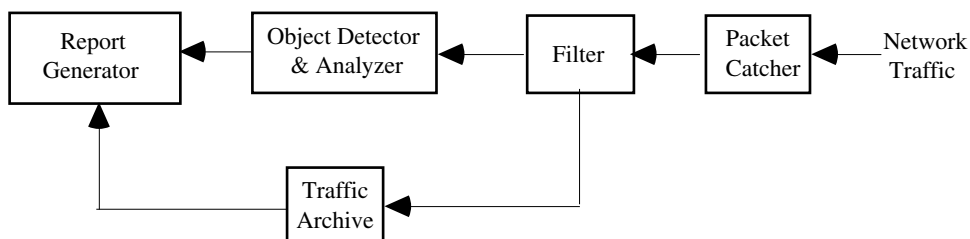


Figure 7.4. NSM Architecture [7.6].

The challenge for NSM is in constructing a picture of what individual users are doing from the numerous individual captured packets. NSM takes a layered approach to this problem. The bottom most layer being the individual packets in the form of a bit stream. The second layer is the thread layer which forms the packets into unidirectional data streams associated with specific hosts and organized into thread vectors [7.9]. The next layer takes the thread vectors and attempts to pair them with another thread vector, representing the bi-directional (i.e., host-to-host connections) nature of network communication. This pairing forms what is referred to as a connection vector. The fourth layer entails the development of host vectors which use the connection vectors to draw conclusions about what each host is doing.

The host and connection vectors are used as inputs to an expert system to analyze the network traffic. The expert system also uses several other types of information to determine if intrusive activity has occurred. The first is a profile of expected traffic behavior. This information consists of, for example, which data paths are expected to occur (i.e., which hosts normally connect to which other hosts) using what type of service (e.g., *telnet*, *ftp*, *mail*, etc.). This expected behavior is based on the past performance of the network. The expert system also knows what level of authentication is required for each service and the level of information that can be obtained using that service. For example, *ftp* does not provide as much capability as does *telnet* but also requires less authentication. All of this information is then compared with signatures of past attacks to ascertain whether particular connections appear to be abnormal and indicate intrusive activity.

NSM was extensively tested with some very interesting results. During a period of two months at UC-Davis, NSM analyzed in excess of 110,000 connections, correctly identifying over 300 intrusions. More significantly, only about one percent of these intrusions had previously been noticed by the system administrators [7.9]. This would seem to validate the need for intrusion detection schemes.

7.3.3 DIDS

The Distributed Intrusion Detection System (DIDS) is a project sponsored jointly by UC-Davis, the Lawrence Livermore National Labs (LLNL), Haystack Laboratory, and the US Air Force [7.9]. It is an extension of NSM and is designed to take care of several deficiencies with NSM. The first deficiency is that NSM does not have the ability to monitor the actions of a user directly connected to a system on the net either through the console or via dial-up lines. Secondly, DIDS is designed to allow intrusion detection activities to take place even in the face of encrypted data traffic (a problem which handicaps NSM). DIDS consists of three components as depicted in Figure 7.5.

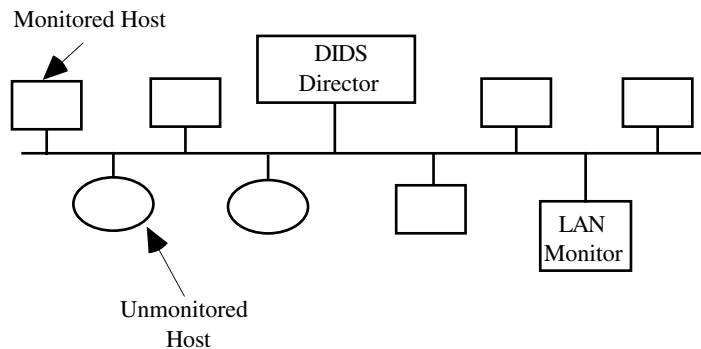


Figure 7.5. DIDS Components [7.13].

The individual **Host Monitors** scan their own system's audit trails looking for events which may indicate intrusions as well as other events which are important to the network. These other events include such activities as network access attempts using *rlogin* or *rsh*. The individual monitors also employ the same algorithms used by Haystack to form their core intrusion detection scheme [7.9]. All important events occurring on the individual hosts are reported to the **DIDS Director** by the **Host Monitors**. The function of the Director is to coordinate the network's intrusion detection activities. The Director communicates with both the individual host and the **LAN Monitor** to request more information on particular subjects as well as to report the current security state of the network. The purpose of the **LAN Monitor** is to observe all traffic on the network and report on network activity such as *rlogin*, *telnet*, and *ftp* connections. Currently, the **LAN Monitor** is a subset of NSM [7.9, 7.13].

Another interesting aspect of DIDS is its ability to track users as they travel across the network. DIDS assigns a unique Network-user IDentification (NID) to all

users as soon as they enter the monitored environment. All subsequent actions by an individual user are then associated with the NID and not a userid or name. In this way, DIDS knows that actions performed by *tom@host1* are to be credited to the same individual as *mary@host2*, even though the user has changed names on the individual hosts [7.9, 7.13]. An even more interesting problem is to track users as they travel through unmonitored hosts. It is a difficult matter to know when user *tom@monitoredhost1* accesses *unmonitoredhost2* and then uses it to log into another system as *mary@monitoredhost3*. A possible solution is to compare packets traveling between hosts. The data portion of packets should be the same for the same individual no matter how many different hosts exist in the individual's trail. To do this, however, involves careful timing and comparison functions and is currently not performed by DIDS.

7.3.4 NADIR

Designed for the Los Alamos National Laboratory's Integrated Computing Network (ICN), the Network Anomaly Detection and Intrusion Reporter (NADIR) is an expert system based anomaly and misuse detection system [7.7]. NADIR takes audit records from the network and generates weekly summaries of network and individual user activity. The associated network, ICN, is not a general purpose network but is tailored to the organization. As such, NADIR is not immediately transferable to other networked environments.

The ICN is divided into four partitions, each processing at a different security level [7.7]. Any workstation connected to a partition can access other systems in its own partition or in a partition at a lower security level. The partitions are linked by a system of dedicated service nodes which perform such functions as access control, file access and storage, job scheduling, and file movement between partitions [7.7]. The audit records used by NADIR come from these service nodes. NADIR compares the summary of the weekly audit data with a series of expert rules which describe security policy violations and suspicious activity. Each user has a 'level-of-interest' value (initially set to zero) which indicates the level of suspicion that an intrusion or some other anomalous activity has occurred with that account. Each time a rule in the expert system matches the profile generated from the week's audit data, the level-of-interest is increased. The amount this value is increased depends on the importance of the matched rule. At the conclusion of the analysis process, a report is generated which lists all users whose level-of-interest value has exceeded a predefined threshold. The rules used in NADIR were developed by interviewing security auditors and administrators tasked with enforcing the laboratory's security policies.

7.3.5 CSM

The Cooperating Security Manager (CSM) is an intrusion detection system designed to be used in a distributed network environment. Developed at Texas A&M University, this system runs on UNIX based systems connected over any size network. The goal of the CSMs is to provide a system which can detect intrusive activity in a distributed environment without the use of a centralized director. A system with a central director coordinating all activity severely limits the size of the network. Instead of reporting significant network activity to a central director, the CSMs communicate among themselves to cooperatively detect anomalous activity. The way this works is by having the CSMs take a proactive, instead of a reactive, approach to intrusion detection.

In a reactive approach with a centralized director, the individual hosts would report occurrences of failed login attempts, for example, to the central director. In a proactive approach, the host from which the intruder was attempting to make the connections would contact the systems being attacked. The key to having a proactive approach work is to have all hosts (or at least the vast majority of hosts) on a network run a CSM. While this may at first appear to be somewhat idealistic, with the current interest among some vendors to develop 'security on a chip', the concept of an intrusion detection system provided in hardware may not be that far off.

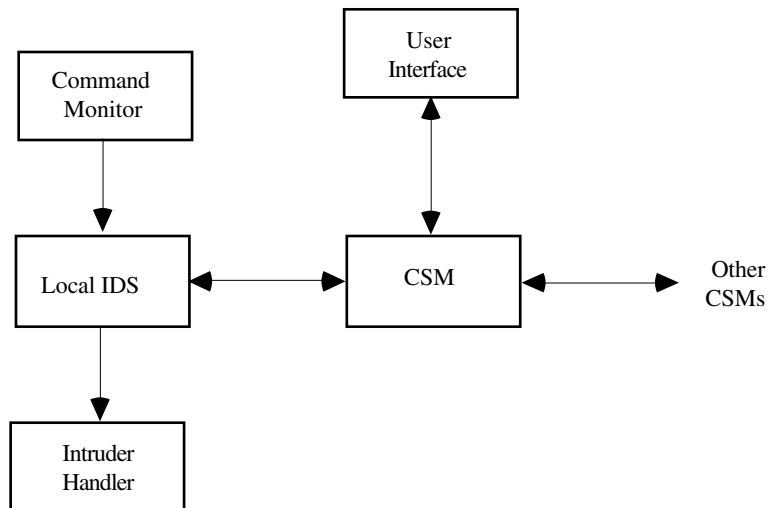


Figure 7.6. CSM components.

The CSMs consist of five components as illustrated in [Figure 7.6](#). The **Command Monitor** is responsible for capturing the commands issued by the users and sending them to the **Local IDS**. The **Local IDS** is in charge of intrusion detection for the local CSM host. Messages relating to important network activity (commands such as *login*, *ftp*, and *telnet*) are sent to the **CSM**, whose job it is to coordinate between the other CSMs on the network. Whenever a connection to another host is made, both CSMs keep track of the connection and any action on the one host is also considered paralleled on the other. This keeps an intruder from attempting a few attacks from one host, and then moving to another host before suspicion has been raised. If a host determines that intrusive activity is occurring, the CSM sends a message to the other hosts in the user's chain of connected systems, to notify them of the incident. Each of these CSMs can continue this notification if the individual originated from still other hosts. The ultimate result is that all involved hosts are notified of the intrusive activity.

The **User Interface** is used by system or security administrators to query the CSM on the current security status of the host. The administrator can query the CSM on individual user's origin and trails, and can request the current level of suspicion for any user. The **Intruder Handler** is designed to take appropriate action when an intrusion is detected. The first course of action is to simply notify the administrator that an intrusion is occurring. Since modern computer systems no longer require constant operator care, there is a good chance that the administrator will not be available when an attack occurs. In that case, the CSM will take further action, if the activities of the intruder are viewed as destructive. These actions include terminating the current session and locking the account.

7.4 Monitoring and the Law

An issue that remains is the legality of monitoring actions of users on computer systems. At first it may appear that an organization has the right to monitor its own computer systems in order to protect itself from damage that could be caused by outsiders or disgruntled employees. The issue, however, is not as clear to the courts and there have been few cases related to monitoring to help set any consensus in terms of legal precedents. Consider, for example, a 1992 related case ruled on by the Pennsylvania Supreme Court dealing with a new telephone service, then being introduced, called **Caller*ID**. This service identified the caller to the receiver of a telephone call. The court ruled that, since *both* parties had not consented to the disclosure of the callers identity, the service violated the federal law and 1988

amendments to the Wiretap Act [7.1]. Basically, current law prohibits surveillance without obtaining a court order with only a few exceptions. One of these exceptions is for individuals to record communication they are part of, with a few states requiring both parties to consent to the recording. In an earlier case in 1978, directly related to computer monitoring, a Court of Appeals ruled that keystroke monitoring did not constitute an illegal wiretap, since it was performed by private individuals and not by a law enforcement agency [7.1].

So what does all of this mean for intrusion detection systems? With so few rulings it becomes essential for those wishing to monitor their systems to take precautions against possible future suits. In December of 1992, the Computer Emergency Response Team (CERT) issued advisory CA-92:19 which addressed the Department of Justice advice on Keystroke Logging. The advisory basically stated that organizations that intend on monitoring their systems should include in the system banner, at logon, a message that states that monitoring will occur, and that anyone using the system, by the mere act of using the system, consents to this monitoring and its use. It is not enough to notify all authorized users when their initial account is set up, it must be displayed at each login, so those not authorized access to the system can also view it. This may actually result in the side effect of scaring off potential intruders, since they know their actions will be monitored.

7.5 Summary

Audit trails which record various actions performed by computer users have been part of modern operating systems for a number of years. The audit trails, which were first used for accounting purposes, were discovered to also have the potential to be used to detect intrusive activity on the computers they monitored. The problem with using the audit trails, however, is that they produce too much information for security managers to effectively handle. The development of a variety of Intrusion Detection Systems, however, has made the use of audit trail information possible.

There are several approaches the IDSs use to detect intrusive activity. The most common is a profile based approach in which current user activity is compared with a stored profile of typical activities for that user. If the current actions are not reflective of normal activity for the user, it may indicate somebody else is using the account. Another approach is action based in which a user's actions are examined for specific actions known to be indicative of intrusive activity. A third approach, which currently is not used due to a lack of available data, is intruder profiling. In this approach, instead of defining typical user behavior, typical intruder behavior is identified. If the current user's activities match the pattern of a typical intruder's, then intrusive activity is indicated. A fourth technique is signature analysis. This method uses physical

characteristics such as typing speed and/or common errors made to authenticate user identity.

A number of IDSs have been developed using a variety of these techniques. Several are designed to operate in a networked environment. Most, however, are still under development, with only a couple being commercially available.

7.6 Projects

- 7.1 For a non-PC based system of your choice, describe specific actions that you would look for to detect attempted/successful break-ins of your system and masquerading. How are these two different? How are they the same?
- 7.2 List several similarities and differences in using audit trail data to detect unauthorized activities performed by insiders and intrusive activities performed by intruders. Which do you think is harder to detect? Defend your answer.
- 7.3 All of the four discussed methods to detect intrusive activity have their strong and weak points. Which of the four do you feel has the most merit? Why? If a combination of methods would work best, what combination would you choose? Defend your choice(s).
- 7.4 One of the most difficult problems in implementing a profile based system is identifying the attributes or measures which enable the system to positively identify a user. Which attributes or measures do you feel provide the best indicators of individual activity? Defend your choice(s).
- 7.5 The problem of tracking users on a network, as they travel between a combination of monitored and unmonitored hosts, is a tricky one. A method proposed by the developers of DIDS is to compare packets. List some of the problems associated with implementing this solution and how you would go about resolving them. (Hint: Consider problems associated with a lack of a global clock as well as problems associated with the splitting up of packets).
- 7.6 Three responses mentioned to handle intrusive activities were to notify the system/security administrator, terminate the session, and lock the account so the individual can not log back in. What are the advantages and disadvantages of each of these responses? Describe other responses that

could be used to limit the damage an intruder can cause while not interfering with authorized users.

7.7 References

- 7.1 Cheswick, William R. and Bellovin, Steven M., Firewalls and Internet Security, Addison-Wesley, Reading, Massachusetts, 1994.
- 7.2 Denning, Dorothy, “An Intrusion-Detection Model”, *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, Oakland, California, April 1986, pp.118-131.
- 7.3 Denning, Dorothy, “An Intrusion-Detection Model”, *IEEE Transactions on Software Engineering*, Vol. SE-13, No. 2, February 1987, pp. 222-232.
- 7.4 Ferbrache, David and Shearer, Gavin, UNIX Installation Security & Integrity, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- 7.5 Gaines, R. Stockton; Lisowski, William; Press, James S. and Shapiro, Norman, “Authentication by Keystroke Timing: Some Preliminary Results”, RAND Technical Report, R-2526-NSF, May 1980.
- 7.6 Heberlein, L.T.; Levitt, K.N. and Mukherjee, B., “A Method to Detect Intrusive Activity in a Networked Environment”, *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 362-371.
- 7.7 Hochberg, Judith; Jackson, Kathleen; Stallings, Cathy; McClary, J.F.; DuBois, David and Ford, Josephine, “NADIR: An Automated System for Detecting Network Intrusion and Misuse”, *Computers & Security*, Vol. 12, No. 3, May/June 1993, pp. 235-248.
- 7.8 Lunt, Teresa; Jagannathan, R.; Lee, Rosanna; Listgarten, Sherry; Edwards, David L.; Neumann, Peter G.; Javit, Harold S. and Valdes, Al, “IDES: The Enhanced Prototype, A Real-Time Intrusion-Detection Expert System”, SRI International Report, SRI-CSL-88-12, October 1988.

- 7.9 Mukherjee, Biswanath; Heberlein, L. Todd and Levitt, Karl, "Network Intrusion Detection", *IEEE Network*, Vol. 8, No. 3, May/June 1994, pp. 26-41.
- 7.10 National Computer Security Center, A Guide To Understanding Audit in Trusted Systems, NCSC-TG-001, Version 2, June 1988.
- 7.11 Sebring, Michael; Shellhouse, Eric; Hanna; Mary E. and Whitehurst, R. Alan, "Expert Systems in Intrusion Detection: A Case Study", *Proceedings of the 11th National Computer Security Conference*, Gaithersburg, Maryland, October 1988, pp. 74-81.
- 7.12 Smaha, Stephen, "Haystack: An Intrusion Detection System", *Proceedings of the IEEE Fourth Aerospace Computer Security Applications Conference*, Orlando, Florida, December 1988.
- 7.13 Snapp, Steven R.; Brentano, James; Dias, Gihan V.; Goan, Terrance L.; Heberlein, L. Todd; Ho, Che-lin; Levitt, Karl N.; Mukherjee, Biswanath; Smaha, Stephen E.; Grance, Tim; Teal, Daniel M. and Mansur, Doug, "DIDS (Distributed Intrusion Detection System) – Motivation, Architecture, and an Early Prototype", *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 167-176.

7.8 Extended Bibliography

- 7.14 Banning, D.; Ellingwood, G.; Franklin, C.; Muckenhirn, C. and Price, D., "Auditing of Distributed Systems", *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 59-68.
- 7.15 Clyde, Allan R., "Insider Threat Identification System", *Proceedings of the 10th National Computer Security Conference*, NIST, September 1987, pp. 343-356.
- 7.16 Dowell, Cheri, "The COMPUTERWATCH Data Reduction Tool", *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 99-108.

- 7.17 Frank, Jeremy, "Artificial Intelligence and Intrusion Detection: Current and Future Directions", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 22-33.
- 7.18 Halme, Lawrence R. and Van Horne, John, "Automated Analysis of Computer System Audit Trails for Security Purposes", *Proceedings of the 9th National Computer Security Conference*, Gaithersburg, Maryland, September 1986, pp. 71-74.
- 7.19 Hansen, Stephen E. and Atkins, E. Todd, "Centralized System Monitoring with Swatch", Technical Report, Electrical Engineering Computer Facility, Stanford University, available via *anonymous ftp* from *mojo.ots.utexas.edu/pub/src/swatch-1.8.6.tar.Z*, March 1995.
- 7.20 Heberlein, L. Todd, "Towards Detecting Intrusion in a Networked Environment", Technical Report CSE-91-23, University of California, Davis, June 1991.
- 7.21 Heberlein, L.T.; Mukherjee, B. and Levitt, K.N., "Internetwork Security Monitor: An Intrusion-Detection System for Large-Scale Networks", *Proceedings of the 15th National Computer Security Conference*, Baltimore, Maryland, October 1992, pp. 262-271.
- 7.22 Heberlein, L. Todd, "Thumbprinting: A Technical Report", Draft Proposal, Department of Computer Science, University of California, Davis, 1994.
- 7.23 Hochberg, Judith G.; Jackson, Kathleen A.; McClary, J.F. and Simmonds, Dennis D., "Addressing the Insider Threat", *Proceedings of the DoE Computer Security Group Conference*, Albuquerque, New Mexico, May 1993.
- 7.24 Jackson, Kathleen A.; DuBois, David and Stallings, Cathy, "An Expert System Application for Network Intrusion Detection", *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 214-225.
- 7.25 Jackson, Kathleen, "Management Issues in Automated Audit Analysis: A Case Study", *Proceedings of the 8th European Conference on Information Systems Security, Control, and Audit*, Stockholm, Sweden, September 1993.

- 7.26 Kuhn, Jeffrey D., "Research Toward Intrusion Detection Through Automated Abstraction of Audit Data", *Proceedings of the 9th National Computer Security Conference*, Gaithersburg, Maryland, September 1986, pp. 204-208.
- 7.27 Kumar, Sandeep and Spafford, Eugene, "A Pattern Matching Model for Misuse Intrusion Detection", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 11-21.
- 7.28 Lunt, Teresa F., "Automated Audit Trail Analysis and Intrusion Detection: A Survey", *Proceedings of the 11th National Computer Security Conference*, Gaithersburg, Maryland, October 1988, pp. 65-73.
- 7.29 Marshall, Victor H., "Intrusion Detection in Computers", *Summary of the Trusted Information Systems (TIS) Report on Intrusion Detection Systems*, Unpublished report, January 1991, available via *anonymous ftp* at *csrc.nist.gov /pub/secpubs/auditool.txt*.
- 7.30 Puketza, Nicholas; Mukherjee, Biswanath; Olsson, Ronald, A. and Zhang, Kui, "Testing Intrusion Detection Systems: Design Methodologies and Results from an Early Prototype", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 1-10.
- 7.31 Schaefer, Lorraine J., "Employee Privacy and Intrusion Detection Systems: Monitoring on the Job", *Proceedings of the 14th National Computer Security Conference*, Washington DC, October 1991, pp. 188-194.
- 7.32 Sisbert, W. Olin, "Auditing in a Distributed System: SunOS MLS Audit Trails", *Proceedings of the 11th National Computer Security Conference*, Gaithersburg, Maryland, October 1988, pp. 82-90.
- 7.33 Snapp, Steven R.; Brentano, James; Dias, Gihan V.; Goan, Terrance L.; Grance, Tim; Heberlein, L. Todd; Ho, Che-Lin; Levitt, Karl N.; Mukherjee, Biswanath; Mansur, Douglass L.; Pon, Kenneth L. and Smaha, Stephen E., "Intrusion Detection Systems (IDS): A Survey of Existing Systems and a Proposed Distributed IDS Architecture", Technical Report CSE-91-7, University of California, Davis, February 1991.

- 7.34 Spirakis, Paul; Katsikas, Sokratis; Gritzalis, Dimitris; Allegre, Francois; Androutsopoulos; Darzentas, John; Gigante, Claude; Karagiannis, Dimitris; Putkonen, Heikki and Spyrou, Thomas, "SECURENET: A network-oriented intelligent intrusion prevention and detection system", *Proceedings of the IFIP SEC'94*, Curacao, June 1994.
- 7.35 Winkler, J.R., "A UNIX Prototype for Intrusion and Anomaly Detection in Secure Networks", *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 115-124.
- 7.36 Winkler, J.R. and Landry, J.C., "Intrusion and Anomaly Detection: ISOA Update", *Proceedings of the 15th National Computer Security Conference*, Baltimore, Maryland, October 1992, pp. 272-181.

DAMAGE CONTROL AND ASSESSMENT

In August 1992, the Texas A&M University Supercomputer Center (TAMUSC) was the victim of a well organized Internet-based attack. Over the course of a two week period, intruders gained access to hundreds of systems and established bulletin boards, tool swapping systems, and training grounds for the beginner. The initial impulse of the TAMUSC was to sever the intruder's connections to the various systems and assess the damage. Circumstances, however, led them to passively monitor and record all intrusive activity [8.8]. The activity monitored by the TAMUSC eventually led to the development of a set of state-of-the-art network security tools and improvements to the security of the Texas A&M computing services. Network attacks such as this were not uncommon; in early 1991 an attempt was made to subvert the security of the computing systems at AT&T Bell Laboratories [8.1]. Security administrators at this location decided not only to monitor the intruder, but to compel him into revealing intruder techniques. What made these two attacks different from others that occurred around the same time was the manner in which the intruders were handled. Monitoring provided valuable information not normally available with standard activity logs; information that could assist in identifying, and possibly controlling, the damage inflicted upon the system. These reactions to network intruders gave the security administrators what they needed—a better understanding of a hacker's techniques.

Monitoring and coercing desired intruder activity is not the only means by which an administrator can attempt to control damage inflicted from a security breach. Other common damage control techniques include patching exploited system holes, invalidating compromised user accounts, repairing corrupted files, and increasing the level of user authentication. Each technique has its advantages, and each provides an administrator different solutions to the intruder problem. The selection of the most appropriate techniques to implement depends upon the goals of the security system, the availability and applicability of each solution, and whether the intruder is logged in or already exited the system.

Damage control is only half of the response process. System security administrators must also assess the damage after an attack. This is often difficult as intruders tend to hide their activity by modifying audit logs and other tracking indicators. System administrators must therefore take steps to ensure an accurate account of an intruder's actions can still be made. Together, this damage assessment and the necessary damage control techniques create a comprehensive intrusion response system.

8.1 Damage Control

As mentioned above, damage control techniques are those actions that are taken while an intruder is logged into a system. Since intruders often limit the amount of time they stay logged into any single system, the techniques must be quickly activated and operate in real time. The benefits of effecting any of these techniques are severely hindered, if not useless, once the intruder logs out of the system. Execution speed and response time is therefore of the utmost importance regardless of which damage control techniques are implemented.

Selecting the damage control technique, or set of techniques, to implement will depend largely upon the computing environment under attack and the goals of the administrator. While Texas A&M University and Bell Laboratories were interested in toying with their intruders, most organizations will wish to remove the intruder from their system, as organizations often do not have the time, human resources, and interest required to trifle with an intruder. Furthermore, many organizations do not wish to risk additional system damage by allowing an intruder access to their system. For this reason each organization must plan a standardized course of action when an intruder is detected. Six possible actions are discussed below; no action is such that it cannot be combined with any number of the others.

8.1.1 Inform the Authorities

Once it has been determined that an intrusion is either in progress or has concluded, a system security administrator should be immediately notified. Communications to an administrator should be done either via telephone or in person as opposed to email or any other means of computer communications. Using a computer system to inform the system administrators of an intrusion poses two significant problems. First, the intruder may prevent the communication from reaching the appropriate administrators. Second, the intruder may be able to intercept the communication and possibly prevent it. In either case, an intruder has the advantage of knowing his presence has been detected. At that point, an intruder may

decide to quietly leave the system and avoid being caught or tracked. Most likely, however, the intruder will either ignore the threat or begin to seriously damage the system knowing that access will soon be denied.

Once contact has been made with the security administrator, key information will need to be communicated. The administrator will need to know how the intruder was detected, what the intruder was doing, and what actions the user has performed since detecting the intruder. This information will assist the administrator in tracking the intruder and determining if the intruder has been warned, or is otherwise aware, that they have been detected. Additional information that may seem of little value should also be given to the administrator.

Upon gathering all available information and verifying the presence of an intruder, the administrators may begin to protect the system and its contents. The exact precautions will depend largely upon how the administrators wish to handle the intruder. The administrators may choose to research the intruder's techniques and methods by logging all activity. Many businesses and professional organizations do not wish to burden themselves with an intruder and will work to remove the intruder. Often the time and resources required to play with an intruder are too costly and yield little results. Additionally, the protection and assistance the legal system could provide if an intruder is caught and brought to prosecution is limited because of the current adolescence of the law in this area.

8.1.2 Backup System Data

Once an intruder's presence has been positively identified, sensitive files should be backed up to a safe location. This may not be necessary if backups are made on a regular basis. The objective of this process is to prevent damage to any sensitive information within the system. Depending on how long the intruder has been able to access the system, this may be useless as the data may have been altered at an earlier time. Nonetheless, backups will be useful in the event the intruder becomes more hostile. This may occur if the intruder believes they have been noticed or becomes frustrated by an unsuccessful or unfruitful attack. It is therefore important to make the backups with as little fanfare as possible.

To prevent the intruder from becoming anxious and modifying or deleting files, the backup process should be performed covertly. This can be accomplished by using accounts not normally associated with system administrator privileges to back up files. These backups should be performed in small, inconspicuous steps as not to attract unwanted attention. Voluminous amounts of file duplication or data migration by an administrator may serve to warn an intruder that a problem is suspected by the administrators. Furthermore, activation of an administrative account will most likely scare the intruder from the system. If an automated (scheduled) backup process can be

activated, it may assist in hiding the administrator's activity. Depending on the sophistication of the intruder, the utility programs that will be used to perform these actions may have been modified. All actions performed when an intruder has compromised a system should be carefully observed as they may not perform their intended task or may be modified to act as a Trojan horse. The possibility of this occurring can be better assessed at the time of the attack.

After the process is verified to properly function, but before the backup process begins, an unused backup medium should be prepared for use. The benefit of using a new medium is threefold. First, it prevents corruption or damage to existing backup data. This provides extra protection against corruption to what may be the only available data for restoration. Second, it provides an independent set of data to use for comparison to other, non-modified data. It may be useful to compare possibly corrupt data to that which is known to be correct and accurate. Third, it begins the logging of the intrusion for future analysis. Many organizations, such as the CERT, use all available intrusion information to determine how and why intrusions occur in search of a means to prevent similar attacks.

In the event that the backup process does not go unnoticed by the intruder, it may be necessary to forcibly remove the intruder from the system or disconnect the systems from the network.

8.1.3 Remove the Intruder

Unauthorized user access is never welcome in a computer system. Many organizations whose livelihood relies on the security and integrity of their computer systems and network work to remove any and all intruders. Other organizations, such as universities and research institutes, are interested in learning from the intruder's actions and therefore allow intruders to remain on their system for some time [8.1]. Eventually these organizations decide to conclude their investigations and remove the intruder from the system as well [8.1]. While this is a wise decision, it may also be difficult.

Research has shown that one of the first actions an intruder takes upon gaining unauthorized access is to modify the system in an attempt to retain their access for an extended amount of time [8.2, 8.12]. This is often accomplished by adding new users, obtaining existing authorized user account names and passwords, planting Trojan horses to obtain passwords, modifying the trusted access permissions, or installing back door access paths [8.1, 8.2, 8.12]. Many of these access paths can be quickly identified and patched. All it takes, however, is one unnoticed means of entry for an intruder to get back into the system. In addition, if the original security compromise is not located and patched, any preventative measures taken will be useless. Incomplete patching will result in two undesirable effects, warning the intruder that their presence is known and

indicating to the intruder how unfamiliar the administration is with their own systems. It may also provoke the intruder into causing serious damage to the system. This is not to say that an intruder should be allowed to run free. Every reasonable means necessary should be employed to keep an intruder from accessing a system.

As expected, all of the security holes mentioned above must be patched. Once the files have been repaired or restored, the passwords and trusted system lists changed, and the original security hole found and removed, the intruder will hopefully no longer have access to the system. It can be expected, however, that the intruder will continue to attempt to access the system. If the patchwork was complete, these attempts should fail. Eventually, the intruder will either stop or find another security hole. Hopefully, the intruder will choose to stop before another hole is found.

8.1.4 Contain and Monitor the Intruder

While removing an intruder is the safest action, allowing an intruder to remain can be a valuable research method [8.1, 8.8, 8.12]. Passively monitoring an intruder's actions can yield information about how the intruder compromised the system, what the intruder does to remain undetected, how the intruder further compromises the system, and the intruder's purpose for invading the system. The results of such an undertaking, however, may not produce a worthwhile result. One such incident where monitoring an intruder yielded little useful information is detailed later in the book (see chapter 17). Other similar efforts, however, have shown the value of monitoring an intruder.

Planning and implementing a monitoring system requires the system administrator to take precautions to prevent damage. For example, an intruder should not be allowed to successfully delete or modify system files or unique data. This can be accomplished by either reducing the number of available commands, modifying and hiding the results of command execution and showing the expected results, or by sacrificing a computer to the intruder. Each of these options will require significant human resources and time to implement and support.

The first monitoring technique, reducing the number of available commands, requires the administrator to provide a reason for the reduction and select the commands to implement. If a believable reason for a sudden reduction in functionality is not provided, an intruder may become suspicious. The administrator must also select the commands that will be available to the intruder. The criteria for selecting a command will vary from system to system, but a good beginning is to choose commands that are non-destructive to the system.

The second and more difficult monitoring technique is to alter the effective results of each command so that they only appear to properly function. This can be quite difficult as the execution of various commands may produce contradictory output. This process requires the administrator to carefully determine, and provide, reasonable output.

The third and possibly most reasonable approach is to offer up a sacrificial computer that will be completely restored after the exercise. A sacrificial computer is one whose files and settings can be modified or deleted by an intruder without concern by the owner. The benefits of this approach are that the computer will function as expected with little owner modification other than severing potentially dangerous network connections. Since the full complement of commands and most network services are available, an intruder's actions will not be limited. Thus, the audit logs will yield a more accurate chronicle of how an intruder operates. The difficulty with implementing this approach, however, is that it requires at least one additional computer. While all of the monitoring methods may require additional printers, storage devices, or other peripherals, this method requires a complete computer. For many organizations, this may be difficult to obtain, thus, forcing an alternate response.

8.1.5 Lock Stolen User Accounts

Most intrusions begin with the compromise of a single user account. This account may be either a system default account, such as *guest*, or an authorized user account. In either case, the administrators should consider locking the account from further use. This will not prevent an intruder from exploiting the account at the time of the attack, but it will prevent further access from the account. This may be a futile attempt to keep the intruder out as other accounts may have been compromised or system vulnerabilities exploited. Additionally, locking an account may not please the account's authorized user. The administrator should consider informing the authorized user as soon as possible so that the denial of service is expected and can be worked around.

This process does not require a complete denial of service to the authorized user. The administrator may choose to provide the user with a replacement account or simply change the password of the existing account. This decision will depend upon the severity of the attack, the needs of the administrator, and the needs of the effected user. If the administrator wishes to monitor the account or fears that it may have been further altered or compromised, the authorized user should be given a new account. In the event that the administrator plans to remove the intruder, and the risk of additional account compromise seems minimal, a simple password change may suffice.

8.1.6 Require Additional Authentication

As mentioned above, circumstance may be such that locking the account will only inconvenienced the authorized user. To further slow down the activity of an intruder, and possibly halt it altogether, the system may require the current user (an intruder) and every subsequent user to provide additional authentication information to

continue to function. For example, a system may require an additional password be provided before file removal or modification is performed.

The additional authentication used for the protection scheme is based upon either a smart-card or calculator, or information provided by a user at the time an account is established (much like the questionnaire based authentication discussed in chapter 5). This will prevent the intruder from taking advantage of any captured authentication information.

While the process of maintaining and using additional authentication may be inconvenient to authorized users, it does not completely prevent them from using the system. The additional authentication, when activated by the system, provides an authorized user with the knowledge that their account may have been compromised. Because the additional authentication is only employed after an account has been compromised, its presence will also signal an authorized user that their data was stolen. This notification will inform a user to take the appropriate actions.

8.2 Damage Assessment

While damage control concerns the response to an attack in progress, damage assessment concerns the post-intrusion/post-mortem of the attack. More precisely, damage assessment refers to the two stage process of first identifying and recovering from an attack, then preventing future attacks. The identification and recovery stage includes such tasks as examining log files, identifying stolen information and user accounts, identifying and locating modifications to both system and user files, and restoring the system to its pre-attack state. Once these actions have been performed, the prevention process may ensue. This process includes searching audit logs for vulnerabilities, patching existing security holes, locking unused and stolen accounts, changing passwords, implementing shadow password files, possibly reducing system services, and creating a clean system backup. As can be seen, many of these actions are performed in both stages of the damage assessment process. Because much of the processes in each stage overlap, the stages are often performed simultaneously, as one indistinguishable process.

The complete damage assessment process does not require a system free from intruders, however, it is best applied when intruders are not logged into a system. An intruders presence would only be destructive to this process because the intruder may actively work to thwart an administrator's actions. The implementation of any damage assessment technique would therefore be best applied after the affected systems were placed into some form of single user mode. If the computer cannot be effectively removed from service, or more than one computer has been compromised, an alternate solution would be to restrict remote access. Even though this will only allow local

users to continue their work, it may be more acceptable than denying service to everyone.

8.2.1 Attack Recovery

The primary goal in the recovery process is to restore all effected systems to, at least, their pre-attack condition. This can be accomplished by either completely reinstalling all software and data or examining each system to locate and correct all intruder modifications. The former process may be difficult if the system was never backed up or it must remain on-line. In that case, the latter process will begin, starting with examination of the audit trails and log files. The audit trails may indicate how the intruder initially gained access and may also indicate what the intruder did once inside the system. The knowledge gained by the examination of the audit trails will assist in identifying stolen or unauthorized accounts. This may not be enough, however, so password file comparisons and additional examination will be required. After the accounts and passwords have been restored to their original state, the system must be examined for unauthorized modification to files, both system and user. This will include searching for such things as Trojan horse programs and unauthorized network services. Once the modifications are located and studied, they must be removed. Upon completion of this process, the system, theoretically, should be completely restored to its original state. Each of the steps in this process is discussed in detail below.

8.2.1.1 Examine Audit Trails

The first step in the recovery and restoration process is the examination of audit trails and log files. Audit trails are on-line records of every command that has been executed on a system. Audit trails were originally intended for computer usage accounting and billing purposes, they have since become a valuable tool in identifying the activities of system users, particularly unauthorized users [8.3, 8.10]. For this reason, they are the first items an administrator should examine after a successful attack has occurred.

Examination of audit trails often yields valuable information concerning the activity of an intruder. They may indicate the type of information an intruder desires, the steps an intruder took to obtain system administrator privileges, and how the system was modified to provide easy access in the future [8.3, 8.10]. [Figure 8.1](#), below, is a segment of an UNIX audit trail where an intruder is attempting to obtain administrator privileges. This log excerpt comes from an audit trail that records all attempts to become the administrator. In this example, the person on user account 'smith' was attempting to obtain administrator status and was finally successful on their fourth attempt. The administrator will have to determine if this was an authorized

user having difficulties logging in or if this was truly an intruder. Examination of other audit logs and discussions with the authorized user of the 'smith' account will help to answer this question.

```
smith *UN*successfully su'd to *ROOT* on Thu Oct 22 00:25:03 CST 1998
smith *UN*successfully su'd to *ROOT* on Thu Oct 22 00:25:43 CST 1998
smith *UN*successfully su'd to *ROOT* on Thu Oct 22 00:26:17 CST 1998
smith      successfully su'd to *ROOT* on Thu Oct 22 00:26:25 CST 1998
```

Figure 8.1. Audit trail records of attempted administrator access.

Directly logging in as the administrator is not the only way an intruder may attempt to obtain additional system privileges. Another means of obtaining administrator privileges is exploiting system vulnerabilities such as a *sendmail* or *CGI* hole. Audit trail records of these activities may indicate the accounts involved in the unwanted and overtly hostile actions. Further examination of these accounts and their activities would be an excellent beginning in the search for other hostile behavior and system modification.

Once an account is suspected of being stolen and exploited by an intruder, the account's activity should be examined. The actions recorded by the system will often indicate how an intruder further compromises security. For example, an intruder may create additional user accounts for future access. Depending upon the configuration of a system's auditing services, the logs may then indicate that the user account and password files were modified with an editor. In some instances, the auditing services may record less. The auditing services can also record modifications made to system files and commands. For example, a common intruder activity is to modify the system login program to record a copy of an authorized user's password. While the recording process may not be apparent, the modification of the login program may still be identified in the audit logs.

Once intruders establish continued access to a system, it is not uncommon for them to either look for valuable data or attempt to gain access into other systems. Both of these tasks are accomplished by searching files, directories, and personal mail. Like file modifications, these actions may also be recorded in the audit logs. Examination of the commands executed by an intruder may reveal their subject of interest. This can be useful during a formal investigation or for placing traps.

In addition to data searches, attempted access to remote systems may be recorded in the audit trails. In some cases, use of the *telnet*, *FTP*, *rlogin*, *rsh*, or *rexec* commands will result in the recording of the remote site's name or IP address. This may also provide insight into the type of information an intruder desires. For example, continuous attempts to access various air force and army systems would be a good

indication that an interest in strategic military information may exist. A second reason for having the remote site names is so that their administrators can be warned and a cooperative effort can be made towards capturing or stopping the intruder.

Clearly, the benefits of audit logs are numerous. This fact, however, is also known to many intruders. This is evident by the number of intruders that disable the logging and remove any record of their presence or activity from the existing logs. A system administrator will therefore need to take a proactive role in the prevention of damage to the audit files. One method to prevent damage or modification is to use a write-once-read-many (WORM) recording medium. Using specialized equipment, however, is often too cost prohibitive for many organizations. A less effective alternative such as audit backups is the next best course of action.

8.2.1.2 Identify Stolen Accounts and Data

One of the benefits of audit logs is the assistance they provide in locating stolen accounts or information. As mentioned above, audit trails may capture an intruder's actions and identify password theft, or the threat thereof. More precisely, this is accomplished by looking for suspicious actions that could be associated with password or information theft. Common actions that indicate accounts are being attacked include the modification, downloading, mailing, or displaying of the password file or other files containing passwords (i.e., a UNIX .netrc file or Microsoft Windows .pwl file) in part or in whole. Unequivocal confirmation that a user is stealing information, however, is more difficult to obtain. The only way to identify stolen information is to examine the search patterns of an intruder. For example, if an intruder is able to locate and review information regarding defense contractors, it could be easily misconstrued that the intruder is most likely to download similar information. Unfortunately, assumptions based upon an intruder's activity are not always correct. That same intruder could just as easily be looking for recreational entertainment. This process clearly requires a thorough examination of the available audit logs.

The benefits of audit logs are no doubt great, however, they are not always available. Attempting to locate stolen passwords or information without the aid of audit logs requires an advanced knowledge of user activity and system operation. This knowledge will be required to locate suspicious account and system activity. Suspicious activity that should be investigated includes accounts that appear unusually active after long periods of dormancy, passwords that are easily determined with the latest password cracking programs, and abnormal account usage times and patterns. Since an intrusion has already occurred at this stage, this information will help to identify accounts that have possibly been compromised. It should be noted that this type of analysis is not appropriate for identifying the unverified existence of an

intrusion; its value is limited to a starting point for an informal investigation of an existing attack.

Conspicuously missing from the list of suspicious activity is abnormal behavior from the administrator (root) account. Much of what is considered abnormal activity by a user's standards is common to an administrator; abnormal administrator activity is difficult to generalize and define and thus difficult to detect. Furthermore, the administrator account should always be assumed to be stolen. The additional privileges that come with the administrator's account makes it a common and valuable target. Additionally, the fact that the audit logs are either erased or no longer being updated should raise concern and suspicion that the administrator's account has been stolen.

Unlike stolen passwords, stolen information is difficult to detect without the assistance of audit logs. When information is stolen, untraceable copies are created or downloaded with little record that any activity took place. Logic and reason will therefore be required to determine what may have been stolen. Administrators will have to determine which files and data the intruder may have accessed. The accessible information will have to be reviewed to determine what would be of value or interest to an intruder. For example, information about last weekend's company softball game will undoubtedly be of little value to an intruder. Scientific research, however, may be of value to an intruder and should therefore be suspected of being stolen. This does not necessarily mean that the information was stolen; simply that it was an easy and possibly attractive target to the intruder. Unfortunately, educated guesses are the only way to determine what information could have been stolen.

8.2.1.3 Locate System Modifications

Equally important to locating stolen accounts and information is identifying unauthorized system modifications. These modifications are often performed to provide an intruder with either an additional means of accessing the system or a safer means of accessing the system. Common modifications to search for are listed in [Table 8.1](#). This list is not intended to be complete, every day this type of information changes and the list grows. Each system will have different traits and vulnerabilities that provide different opportunities for an intruder.

Each of the modifications listed in [Table 8.1](#) provides an intruder the means of returning to the system without detection or with what appears to be a valid user account. Unauthorized and unnoticed access such as those listed will invariably lead to the compromise of the system administrator's account.

Table 8.1. Possible intruder installed system modifications.

- Modification to the login program
- The introduction of old, presumably patched vulnerabilities
- Addition or modification of network services
- Addition or modification of automated system tasks
- Addition or modification of user accounts
- Modification of system shells
- Addition of trusted users or systems
- Removal or modification of system auditing services and files

Locating the types of modifications listed in [Table 8.1](#) proves to be a straight forward task when assisted by audit logs. Audit logs are only of limited assistance, however, as at best they can only indicate what files or programs were modified, not how they were modified. Additionally, they may have been modified to hide intrusive behavior. A knowledgeable user will therefore be required to scrutinize the files and locate all unauthorized changes. Locating every modification is a difficult and imprecise process even with an expert knowledge of the system. As can be expected, locating newly installed holes becomes even more difficult when audit logs are unavailable.

Without the assistance of audit logs, the administrators will need to thoroughly search the system for newly placed vulnerabilities that may not even exist. One means of doing this is the examination of file modification dates. Files that have recently been modified without a legitimate reason should be more closely examined. Most intruders, however, are quick to restore any tell-tale file modification dates as not to attract unwanted attention to their access paths. Laboriously examining every file for modifications could be greatly simplified, however, if some precautions were taken before the attack. One such precaution is the calculation of file checksums. File checksumming is the process of mathematically relating, or *hashing*, large amounts of data, usually a file, down to a simple, significantly smaller, and more manageable value [8.6, 8.7]. As long as the original data remains unchanged, the checksum value will remain unchanged [8.6]. In the event that the original data is modified, the related checksum value will change. Thus, if a file's original checksum value does not match a newly calculated checksum value, the file has been changed. Some people may question whether modifications can be made without a change appearing in the associated checksum value. While some early checksum algorithms may have produced the same checksum value for both the original data and the modified data, newer checksum algorithms have made this near impossible [8.10]. The benefits of the checksum process are nonexistent, however, if a trusted set of checksums were not previously created and updated.

8.2.1.4 System Restoration

Once files have been identified as needing replacement or recovery, original files should be loaded into the system. If at all possible, these replacement files should come from the vendor provided distribution disks. The administrator should also verify that any subsequently released patches are also applied. Failure to do this could result in the removal of one set of vulnerabilities at the cost of introducing an older, more widely known set of vulnerabilities. In the event that the distribution copies of files do not exist, as is the case with the user account and password files, the most recent, and unaltered, set of files should be reinstalled. This too may result in the reintroduction of vulnerabilities, such as compromised passwords. These holes, however, should have been previously identified and easily corrected. In the event that an unmodified version of a file does not exist, the intruder-modified version will have to be repaired. This should be rare, however, as many systems follow a rigid and periodic file backup process.

8.2.2 Damage Prevention

Once a system has been restored to its pre-attack state, or before an attack ever occurs, action should be taken to prevent future damage and system compromise. This can be accomplished with a few simple techniques. These techniques are not intended to completely protect a system from attack and compromise, rather, they serve as a minimalist effort that should prevent a large number of possible system break-ins. The six security techniques that are discussed below improve the posture of a computer system and should always be enforced. These techniques are: patch security holes, lock stolen user accounts, change account passwords, employ shadow password files, backup information, and reduce network services. They are not intended to compromise a complete and thorough security system—such a system would include many of the other concepts discussed throughout this book, such as firewalls, authentication systems, and encryption.

8.2.2.1 Patch Security Vulnerabilities

Previous audit logs and system examinations should reveal weaknesses in a system. These weaknesses, whether maliciously applied or previously existing, must be patched. Many will be well known and therefore well documented. Other system vulnerabilities will be new, and possibly unique to a particular system. In either case, the patches should be applied and documented. This will not only protect the system from future attacks, but this will also provide a record of patches and modifications that have been applied or need to be applied.

A list of previously applied patches and vendor recommended patches will be helpful when reinstalling a new copy of the operating system or just verifying the presence of patches. These lists will help ensure that all known system vulnerabilities are again protected and no longer pose weaknesses in the system. It is easy to forget one or two special patches or system modifications when a system is reinstalled. Even though many systems are configured to install many existing patches, the documented lists will guarantee that site-specific modifications are applied as well. Furthermore, the lists will also serve the purpose of providing a distinct checklist of patches to examine after an attack. It cannot be guaranteed that every previously installed patch will exist, or even function after an attack. Many intruders remove or modify existing patches because it is not so easily noticed by administrators. Administrators often wrongly assume that existing patches still exist and function properly after system compromised. Administrators should therefore verify that a patch not only exists but properly functions by thoroughly testing or reinstalling it.

8.2.2.2 Lock Stolen User Accounts

Stolen user accounts are a valuable tool to an intruder. They provide an intruder the means of accessing a system under while appearing to be an authorized user. For this reason, system administrators should attempt to keep intruders from stealing user accounts and masquerading as authorized users. One way to reduce the vulnerability of user accounts is to work with users in controlling their accounts. Users and administrators must keep each other abreast of changes in an account's activity and privileges. Users must inform administrators of times when their accounts will be inactive and discuss their needs and usage patterns. Similarly, administrators should safely provide the resources required by a user, but also monitor unusual activity and restrict the privileges allotted to an account. When administrators and users work together, stolen accounts can be prevented and quickly detected.

One of the greatest problems in locating stolen accounts and preventing their theft is the inevitable breakdown in communication between administrators and users. Administrators must be informed of changing account usage patterns as soon as possible. This will allow the administrators to provide a continued level of high performance in a transparent fashion. Changes worthy of administrator notification include remote-only access when traveling, temporary inactivity such as when on vacation, and permanent inactivity such as when changing employers. This notification will allow administrators to keep unauthorized usage to a minimum while providing authorized users the resources and security they require.

Despite poor communications, administrators should actively search for and lock or remove accounts that have been dormant for a lengthy time. These accounts are attractive to intruders because the authorized users are not likely to notice the

unauthorized use of their account. Additionally, an intruder can hide in a dormant account without raising much suspicion. In an effort to reduce this type of abuse, administrators must strictly adhere to an account locking and removal policy. The policy may simply require users to periodically revalidate their accounts or direct the administrator to lock an account after a predetermined length of inactivity. Unfortunately, this will only provide a limited amount of prevention from stolen accounts. Users will have to provide support by controlling access to their accounts with tools such as password protected screen savers and following generally accepted password usage controls (e.g., minimum password lengths and password expirations).

8.2.2.3 Change Passwords

The concept of changing account passwords after an incident is both simple and obvious. Nonetheless, it must be carefully planned so that the changes are thorough. This is accomplished by taking a two phased approach to password change: (1) protect privileged accounts and (2) protect user accounts.

The first phase, protecting privileged accounts, requires an administrator to change the passwords of all administrative accounts as well as system default accounts. Most people would believe that this would be quite obvious and need not be mentioned. Surprisingly, however, this is a real problem and concern. Experience has shown that many administrators do not change system default passwords, thus giving uncontrolled access to anyone with a passing familiarity of a system [8.12]. To protect a system from abuse, all system passwords should be changed not only after an attack, but on a regular basis. For many organizations, this is standard operating procedure. Even still, attempting to compromise system accounts by using default passwords is a very common, and successful, attack method.

The second phase of protecting a system from attack via compromised accounts is to change the passwords of the users accounts. Clearly, any known stolen account passwords should be changed they are a liability to the security of a system. In addition, all users should be instructed to change their passwords. This will prevent accounts whose passwords are known to an intruder from being attacked. It will also prevent an intruder from accessing accounts that the administration does not realize are stolen. One problem that may result from informing all users to change their passwords is wide spread panic and concern. One way to prevent this is to institute a plan that requires users to change their password on a regular basis. This will force users to change the password they used at the time of an attack, thus invalidating any existing passwords lists obtained by the intruder. The drawback to this plan may be a small backlash from users who do not want to change their password. While serving and assisting the users is one of the tasks of an administrator, the importance of

providing a secure computing environment is a higher priority and should be treated as such.

Once these two phases have been completed, stolen user accounts should no longer be an immediate concern. This is not to suggest that this will not be a problem in the future, rather current vulnerabilities from stolen accounts should no longer pose a significant threat. The need to prevent any further accounts from being stolen, however, is great. One such method to reduce the possibility of account theft is the use of shadow password files.

8.2.2.4 Employ Shadow Password Files

Many existing operating systems, including various derivations of UNIX, store user account information in the same file as the account passwords. This was initially done in an effort to improve system performance and conserve storage space [8.2]. An unfortunate drawback of this data organization is the requirement that the data be made readable to many utilities, and thus readable to every user. Because this allowed anyone to obtain every user's encrypted password, it presented a significant security breach. In an effort to prevent the encrypted passwords from being made public, shadow password files were introduced [8.2].

Shadow password files are access-restricted files that contain the encrypted passwords of each system account [8.2, 8.5]. This enables the encrypted passwords to be removed from the account information file, thus increasing system security without negatively effecting the daily operation of a system [8.4, 8.5]. An example of a shadowed and non-shadowed user information file from a Sun Microsystems Solaris operating system are shown in [Figures 8.2\(a\)](#) and [8.2\(b\)](#). The general information in both [Figures 8.2\(a\)](#) and [8.2\(b\)](#) differ only in that the encrypted password information has been removed from the account information file in the non-shadowed file, [Figure 8.2\(b\)](#). The removal of the encrypted passwords promotes system security as the freely available account information files provide only limited information about the system and its users.

```

mickey:Qh1h7H25GG:1000:1001:Mickey's acct:/home/usr/mick:/bin/tcsh
donald:i9H45d8Sh:1000:1002:Donald's acct:/home/usr/donald:/bin/csh
minnie:hgrD4496a:1000:1003:Minnie's acct:/home/usr/minnie:/bin/csh
daisy:Huyt6G4x8:1000:1004:Daisy's acct:/home/usr/daisy:/bin/bsh

```

(a)

```

mickey:x:1000:1001:Mickey's acct:/home/usr/mickey:/bin/tcsh
donald:x:1000:1002:Donald's acct:/home/usr/donald:/bin/csh
minnie:x:1000:1003:Minnie's acct:/home/usr/minnie:/bin/csh
daisy:x:1000:1004:Daisy's acct:/home/usr/daisy:/bin/bsh

```

(b)

Figures 8.2. (a) Part of a non-shadowed password file.
(b) An equivalent shadowed user information file.

Encrypted passwords are key in retaining access to a computer system. As previously mentioned, once intruders gain access to a system, they work to retain their access and even gaining greater privileges. This is often accomplished by stealing accounts, or more precisely, stealing the encrypted passwords. Once the encrypted passwords are stolen, they may be compared against an encrypted version of a dictionary. The results of which are the plain-text passwords that will probably make up many of the encrypted passwords used to access a system. Preventing an intruder from obtaining a copy of the encrypted passwords may prevent an intruder from gaining access to the system. For this reason, shadow password files are now used and supported by most operating systems and should be seriously considered for all existing systems as well.

8.2.2.5 Backup Information

One of the key goals when recovering from an attack is to restore a system as quickly as possible. This is an issue primarily because the down time after an attack is equivalent to lost computing time and therefore lost money. System administrators should therefore have a means to quickly recovery from an attack. One way to do this is to restore to a clean operating system and then work to recover any compromised accounts and passwords. If periodic system backups are made, this is not a problem. In the event that system administration has been lax and backup files are either out of date or non-existent, recovery will be a more involved process.

Most system administrators, and users, understand the importance of backing up data—many administrators have had to load backup files when a user accidentally deletes their files. Besides protecting users from themselves, backup files may provide a copy of a complete and properly configured operating system. Thus, in the event that a disk fails, recovery is as quick and easy as loading a tape. The same is true for intrusions—if

piecemeal system restoration becomes too great, recovery from a backup file may resolve much of the damage. This only solves part of the problem, however, as the recovered system still contains the same vulnerabilities that allowed the intruder to access the system in the first place. For this to be a viable solution, the system must be recovered to a known, pre-intruder state and the exploited vulnerabilities need to be patched. Once the system is restored and patched, a fresh backup should be made. This backup should also be somehow identified as 'intruder-free' for expeditious recovery in the future.

Not every system has the benefit of backup files. In such a case, a corrupt system can either be completely recovered from distribution files or patched as needed. Because patching is often an incomplete process, it may be in the best interest of all users and administrators to reinstall a system from the distribution files. While this process is neither enjoyable nor quick, it may be more complete than patching an overly stressed system. This process may also be aided by the automatic application of newly available patches that protect a system from similar attacks. This may not result in complete recovery because site specific modifications may still need to be applied—modifications that would be present if recovering from organizational backups as opposed to distribution files. On its own, this process may require extensive efforts for long periods of time. Once this laborious process has completed, the system administrator would be well advised to begin to backup the system on a periodic basis.

8.2.2.6 Reduce Network Services

Many network attacks begin with the compromise or exploitation of existing network services. One such well documented attack exploited the UNIX *finger* command [8.9]. In an effort to reduce intrusion susceptibility from similar attacks, a system administrator may wish to remove any unnecessary or underutilized services. Some such network services that should always be considered for removal are *TFTP* (Trivial File Transfer Protocol), *finger*, *rexec* (remote execution), *rsh* (remote shell), and *rlogin* (remote login). While other services exist and should also be removed, these are excellent candidates as they provide services that are useful but unnecessary for most authorized users. Furthermore, these services often make an intruder's work easier because they are improperly configured.

The Trivial File Transfer Protocol, *TFTP*, is a simple file transfer scheme. This protocol is similar to its more feature-filled counterpart, the File Transfer Protocol (*FTP*). The key differences between the two protocols is that *TFTP* provides no user authentication and is based on a User Datagram Protocol (UDP) as opposed to a Transmission Control Protocol (TCP) [8.11]. These missing features reduce the security provided by the *TFTP* service. One such security shortcoming is *TFTP*'s allowing sensitive system files, such as password files, to be downloaded to a remote

system [8.2]. With the ever present possibility that user passwords are easily determined, this should be prevented. This problem can usually be resolved by altering the execution options of the *TFTP* daemon [8.2]. This alteration will often be necessary as many manufacturers distribute their operating system software with the *TFTP* vulnerability present [8.2].

The *finger* service, much like the *TFTP* service, is also commonly removed. While older holes in the *finger* service allowed commands to be executed remotely have been removed, the information produced by most default *finger* services can be dangerous in the wrong hands. This information often includes the date of each user's last system access, user's account name, and personal information that may be useful when guessing passwords [8.4]. As this information is frequently enough to begin a successful attack on a computer system, many organizations remove the *finger* service [8.4]. In the event that an organization finds value in the *finger* service, a less revealing version of the service may be found and easily installed. Administrators should evaluate the value of the default *finger* service and consider its removal.

Remote access service such as *rsh*, *rexec*, and *rlogin* are also excellent candidates for removal. These services are intended to allow authorized users access to trusted systems without having to provide any password authentication [8.4]. This unchallenged access, however, is not limited to authorized users. Once an intruder has obtained access to one system, the remote services can allow the intruder to access other trusted systems without a password. In an attempt to confine the effects of a system compromise, these service should be denied as well as the *TFTP* and *finger* services. Many system administrators are reluctant to remove these services as they often improve a system's ease-of-use or are necessary for poorly developed (and often home-grown) software to properly function.

The *TFTP*, *finger*, and remote services are worthy of removal when attempting to improve system security. As previously stated, however, they are not the only services that whose presence should be reevaluated. Other services that should be considered for removal are: *biff*, *link*, *netstat*, *NFS*, *NIS*, *sprayed*, *systat*, *tcpmux*, and *uucp*. As is to be expected, the specific services that should be removed from a system will depend upon the needs of the users and the requirements of the organization.

8.3 Summary

Damage control and assessment is the activity of restricting the damage that an intruder can inflict upon a system as well as determining what damage an active intruder has already done. The means of damage control range from simply informing the necessary authorities to monitoring and manipulating an intruder's activity. Many of

the techniques employed for damage control require careful planning so that they can be rapidly and correctly executed. While damage control techniques are being implemented throughout a system, damage assessment procedures can commence. Damage assessment techniques are intended to restore a system to, at least, its pre-attack state. These techniques alone are not enough to protect a system from attack. They are a fundamental set of actions that can be performed to improve the basic security of system. A comprehensive damage prevention program will address such additional issues as firewalls and encryption.

8.4 Projects

- 8.1. Select two or three of the damage controls and implement them. Discuss how the damage controls interact and impact each other.
- 8.2. [Table 8.1](#) lists some possible system modifications that an intruder may attempt. What other system modifications can be made? How will these modifications effect the security and integrity of the system? Demonstrate how they can be detected and prevented?

8.5 References

- 8.1 Cheswick, William R. and Steven M. Bellovin, Firewalls and Internet Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1994.
- 8.2 Curry, David, UNIX System Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1992.
- 8.3 Denning, Dorothy, "An Intrusion Detection Model", *IEEE Transactions on Software Engineering*, IEEE Press, New York, New York, Vol. SE-13, No. 2, February 1987, pp. 222-232.
- 8.4 Ferbrache, David and Shearer, Gavin, UNIX Installation Security and Integrity, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- 8.5 Frisch, Aileen, Essential System Administration, O'Reilly & Associates Inc., Sebastopol, California, 1991.

- 8.6 Garfinkel, Simson and Gene Spafford, Practical UNIX Security, O'Reilly & Associates Inc., Sebastopol, California, 1992.
- 8.7 Russell, Deborah and Gangemi Sr., G. T., Computer Security Basics, O'Reilly & Associates Inc., Sebastopol, California, July 1991.
- 8.8 Safford, David; Schales, Douglas and Hess, David, "The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment", *Proceedings of the Fourth USENIX Security Symposium*, Santa Clara, California, October 1993, pp. 91-118.
- 8.9 Seeley, Donn, "A Tour of the Worm", Technical Report, The University of Utah, available via *anonymous ftp* from *cert.sei.cmu.edu/pub/virus-l/docs/tour.ps*, January 1990.
- 8.10 Stalling, William, Network and Internetwork Security: Principles and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- 8.11 Stevens, W. Richard, TCP/IP Illustrated, Volume 1, Addison-Wesley, Reading, Massachusetts, 1994.
- 8.12 Stoll, Clifford, The Cuckoo's Egg, Doubleday Books, New York, New York, 1989.

8.6 Extended Bibliography

- 8.13 Chantico Publishing Co., Disaster Recovery Handbook, TAB Professional and Reference Books, Blue Ridge Summit, Pennsylvania, 1991.
- 8.14 Hansen, Stephen E. and Atkins, E. Todd, "Centralized System Monitoring with Swatch", Technical Report, Electrical Engineering Computer Facility, Stanford University, available via *anonymous ftp* from *mojo.ots.utexas.edu/pub/src/swatch-1.8.6.tar.Z*, March 1995.
- 8.15 Baker, Mary and Sullivan, Mark, "The Recovery Box: Using Fast Recovery to Provide High Availability in the UNIX Environment", *1992 Summer USENIX Conference in San Antonio, Texas*, June 1992, pp. 31-43.

- 8.16 Blaze, Matt, "NFS Tracing by Passive Network Monitoring", Technical Report TR-355-91, Department of Computer Science, Princeton University, available via *anonymous ftp* from *ftp.dsi.unimi.it/pub/security/docs/papers/nfsspy.ps.gz*, January 1994.
- 8.17 Brand, Russell L., "Coping with the threat of Computer Security Incidents: A Primer from Prevention through Recovery", Technical Report, available via *anonymous ftp* from *csrc.nist.gov/pub/secpubs/primer.ps*, December 1992.
- 8.18 Farmer, Dan and Venema, Wietse, "Improving the Security of Your Site by Breaking Into It.", Technical Report, available via *anonymous ftp* from *ftp.dsi.unimi.it/pub/security/docs/papers/sag_to_crack*, January 1994.
- 8.19 Spafford, Eugene H. and Weeber, Stephen A., "Software Forensics: Can We Track Code to its Authors?", *Purdue Technical Report CSD-TR-92-010*, Department of Computer Sciences, Purdue University, February 1992.

9

DATABASE SECURITY

Database systems effect every person's daily life. They are used in billing systems, subscription services, reservations systems and numerous other aspects of daily life. For these systems to properly carry out their respective tasks, the database operations must function properly and the information contained within the database must be correct. While the data cannot be guaranteed from accidental changes, it can be protected from malicious system and data attacks. This protection is the subject of this chapter. This chapter begins with a database primer followed by a discussion of various database security vulnerabilities and countermeasures. The chapter then concludes with an examination of existing security database models.

9.1 Database Management System Primer

To understand the security problems that can plague a database management system (DBMS), a basic knowledge of database systems is required. A brief discussion of database systems, including an introduction to database terminology, is presented.

Two key parts of a database management system are (1) the user defined data (data tables) and (2) the operations defined to manipulate the data [9.5]. The data tables, often called entities or touples, contain the database information. The tables are composed of constituent characteristics, or attributes [9.5]. For example, the employee data table in [Figure 9.1](#) has four attributes: employee name, address, phone number, and department. In a similar manner, the department data table in [Figure 9.1](#) consists of three attributes: department name, manager name, and location. The data contained in tables such as these are manipulated with operations provided by the database management system (DBMS).

EMPLOYEE TABLE

NAME	ADDRESS	PHONE	DEPT.
Fred	1234 Main Street	555-1234	Testing
Barney	9830 Stone Avenue	555-9830	Sales
Wilma	2838 Quarry Court	555-2838	Testing
Betty	2212 Lime Road	555-2212	Design

DEPARTMENT TABLE

DEPT.	Location	Manager
Design	BLDG. 210	Donald
Payroll	BLDG. 209	Daisy
Sales	BLDG. 236	Mickey
Testing	BLDG. 229	Minnie

Figure 9.1. Example database tables.

Manipulation of the data tables is accomplished with operations that add new data, update existing data, delete existing data, and list data with certain attributes [9.5]. The specific functions that perform these operations depends upon the requirements of the system. For example, a query could request a list of all employees in a given department. Depending on the design and scope of the DBMS, a query such as this may not be possible.

In some database environments the actions available to the users are limited. The limitations can either be the result of the environment or a restriction in a user's capabilities [9.6]. Environmental limitations prevent users from performing actions on data not related to the relevant environment. For example, environmental limitations would prevent a user from modifying their bank records from within an airline reservation system. Unlike environmental limitations, capability limitations restrict the actions a user can perform on relevant data. An example of a capability based limitation would be an airline reservation system that allows the travel agent to change a traveler's reservation but not alter an airplane's flight schedule. Capability based limitations such as this are often implemented with the use of multi-leveled security [9.16, 9.18].

Multi-leveled DBMSs attempt to control access to the system with the use of relative security classifications [9.1, 9.12, 9.21]. Commonly used security labels are top-secret, secret, confidential, restricted, and unrestricted. Security classifications such as these are used to determine whether a user can both access data and perform a given database operation. This is accomplished by applying the security classification labels to the system data, operations and users. Users with a security label lower than that of

an operation are prevented from performing the operation. Users with a security label lower than that of the data are not permitted to read the data. The ability to write information to the database will depend on the access rules enforced by the system. For a user to manipulate data, the user must have a security label that allows the operation to complete on the requested data and access to the data. For example, a user wishing to read a name from an on-line phone book must have the security classification that allows the user to read information from the phone book. The permission to access information in a given manner, such as reading from, or writing to, a phone book is called a role. Roles are discussed in detail below.

9.2 DBMS Vulnerabilities and Responses

DBMSs may suffer from any of four different types of vulnerabilities [9.3, 9.13]. These vulnerabilities are: (1) inference, (2) aggregation, (3) data integrity, and (4) Trojan horses. The presence of inference or aggregation vulnerabilities leads to the undesired release of normally secure, or hidden, information contained in the database. Data integrity vulnerabilities undermine the correctness of the information within the database. Trojan horse vulnerabilities allow hidden operations to perform unauthorized actions in, and to, the DBMS. Removing these vulnerabilities often involves reorganizing the data tables, reclassifying the data, or improving the access controls and general security of the system.

9.2.1 Inference

Inference vulnerabilities in a DBMS allow a user to deduce the contents of inaccessible parts of a database [9.3, 9.12, 9.13]. For example, a database system indicates that a certain jet fighter with a maximum payload of 800 pounds will be flying a mission where it will carry a certain type of bomb. The database system, however, does not state how many of the bombs will be carried. Since it is general knowledge that each of the bombs weighs 100 pounds, it can be inferred that the fighter will carry 8 bombs. This example shows how common sense knowledge may be used to deduce the contents of the database. Inference vulnerabilities, however, may be made with the assistance of not just common sense, but other data within the database or external information [9.3, 9.5, 9.13]. The fact that information used to make inferences comes from uncontrollable sources hinders the protection required to prevent inference attacks.

Properly protecting a DBMS from inference attacks is a difficult task. This is because attackers do not need to perform any overtly malicious actions to determine the

contents of the database and an attacker's presence and activities may go unnoticed. To determine what can be inferred from the system and create an optimal defense, a knowledge of an attacker's source of information is required [9.9]. Since this knowledge is often unattainable, the DBMS must be maintained such that external knowledge provides little assistance to the attacker. Modifying a database so that little can be inferred first requires identifying the information that must be protected and then adjusting the security classification of the data, the appropriate data relations, and operations [9.3]. Naturally, this is an application specific solution and modifications such as these must be made on a case by case basis.

9.2.2 Aggregation

Like inference, the damages that results from aggregation are only information dissemination, not database modification. Aggregation is the process of combining multiple database objects into one object with a higher security label than the constituent parts [9.11]. Combining two objects of similar types, such as two addresses, is called inference aggregation [9.9]. When the combined objects are of different types, such as a name and salary, the process is called cardinal aggregation [9.9]. While the two types of aggregation are similar in nature, they each require different means of prevention.

9.2.2.1 Inference Aggregation

As previously mentioned, inference aggregation is the result of combining the knowledge gained from multiple database objects [9.9, 9.21]. For example, knowing the information contained in any one of the tables in [Figure 9.2](#) provides little information by itself. Knowing the contents of all three tables, however, results in the dissemination of what would otherwise be considered hidden, or private, information.

Inference aggregation can be prevented, by raising the security labels associated with any one or more of the tables involved in the aggregation [9.7]. For example, raising the security label of any single table in [Figure 9.2](#) prevents an authorized user from aggregating the data. This results in requiring a DBMS user to have a higher security label to make the aggregation inference. Raising the security labels of the DBMS tables, however, may result in reducing the operations available to users with lower security labels. The information in a newly labeled data table will no longer be available to lower labeled users who require access. This side effects can be reduce by relabeling only the less frequently used data tables.

PHONE BOOK TABLE

NAME	MAIL STOP	PHONE	DEPT.
Fred	229-234	3-1234	Testing
Barney	236-652	6-8733	Sales
Wilma	229-176	6-2938	Testing
Betty	210-234	3-9090	Design

EMPLOYEE TABLE

NAME	EMPLOYEE	HOME	HOME
	NUMBER	ADDRESS	PHONE
Fred	143324	1234 Main Street	555-1234
Barney	135631	9830 Stone Avenue	555-9830
Wilma	114356	2838 Quarry Court	555-2838
Betty	113542	2212 Lime Road	555-2212

SALARY TABLE

EMPLOYEE	SALARY	RETIREMENT
NUMBER		FUND
938200	82,200	44,000
339483	45,400	22,100
087393	66,100	33,870
947394	52,100	10,790

Figure 9.2. Example of Inference Aggregation.

9.2.2.2 Cardinal Aggregation

Cardinal aggregation is the combining of like information in a manner that yields information not readily available or apparent to a user [9.9, 9.21]. A commonly used example of cardinal aggregation is a phone book. One phone number provides little information besides a name and phone number. A complete phone book, however, can provide information no single phone number can. It is unusual for dissimilar businesses to share a phone number. If a phone book were examined, and a list created of all businesses that share a single phone number, it could be inferred that the listed businesses may be either store-fronts or involved in unusual business practices. Another common example concerns military troop placement information. It may be

of little value to know the location of one army troop. Knowing the location of every troop in the army, however, will be of significantly more value.

This type of aggregation, unlike inference aggregation, is difficult to prevent. Two generally accepted solutions are to: (1) lower the security label assigned to the data and restrict access and (2) raise the security label assigned to the individual data so that it equals that which can be aggregated [9.7]. Unfortunately, neither of these solutions provide a complete solution to the problem [9.21].

The first means of preventing aggregation is the downgrading of information labels and restriction of access. This solution requires a knowledge of both how the data can be exploited and the amount of data necessary to perform an accurate aggregation. It may be difficult, however, to determine, the minimal number of tuples needed to provide additional information by aggregation. Furthermore, attempting to limit the amount of data any user may access at any one time does not resolve the problem as a user may make multiple, smaller database accesses and build a complete database. For example, if a user is limited to requesting only 10 entries from a phone book at one time, the user need only make numerous 10 entry requests until the desired aggregation effect is possible. Clearly this solution does not completely prevent cardinal aggregation, rather it slows down the process by requiring a user to apply more effort to obtain the necessary data for aggregation.

The second cardinal aggregation solution is to raise the security label of each individual data record. Again using the phone book example, each name and phone number would be given a security label equal to that of the aggregated information. Thus, the information gained through aggregation is readily available to a user without aggregation. This solution, however, has two complications. First, it artificially inflates the security labels of the DBMS information. This is evident by the higher security label a user would need to access a phone number in the example above. Second, the highest security level of all possible aggregations must be determined so that the appropriate label can be applied to the data. This requires the ability to determine every possible aggregation from the data so that the proper label can be determined and applied. Determining every possible aggregation is often impossible, making it difficult to guarantee that the appropriate security label is applied [9.7, 9.12].

Unlike inference aggregation, there is no perfect solution to cardinal aggregation. With each solution comes unwanted side effects that make interaction with the database difficult. While some mechanism should be in place to reduce the possibility of aggregation, the security administrator should be aware that any protection will be incomplete.

9.2.3 Data Integrity

Data integrity vulnerabilities allow the unauthorized or inappropriate modification, addition, or deletion of database information. Data integrity vulnerabilities can be prevented with the implementation of an integrity policy. Many policies have been proposed, some of these policies include the Bell-LaPadula policy [9.1], the Biba policy [9.2], and the Schell-Denning strict integrity policy [9.15]. The basic concept behind these policies, and others, is the same: reduce the privileges any user has at any given time and group related DBMS privileges [9.10, 9.19].

The goal of privilege reduction is to limit the operations that any one user has at any given time [9.10, 9.19]. For example, it is unnecessary to give a technician and personnel manager the same DBMS capabilities. A technician need not be able to add new employees or change an employee's personal data. Likewise a personnel manager does not need the ability to modify any technical information. Therefore, DBMSs employ privilege reduction; they only grant each employee access to the data and operations that are required for an employee to perform their job.

Privilege reduction is not limited to one dependency. That is, the privileges granted to a user may depend not just upon their department, but upon their current task as well. For example, one personnel employee may only be able to add employees while another personnel employee may only be able to adjust the position and salary of an employee. This is shown in [Figure 9.3](#). It should also be noted that more than one user may be assigned the same privileges; many users often have the same privileges. For this and other security reasons, DBMSs often employ a means of grouping related privileges called roles.

	Update Records	Remove Employee	Add Employee
Bob	✓	✓	✓
Lisa	✓		
Mark	✓	✓	
Sarah	✓		
Cindy	✓	✓	

Figure 9.3. An example of user privilege mapping.

Grouping related DBMS privileges into a single representation (a role) allows system administrators to better control and represent the activities permissible by a user [9.17]. By creating roles that represent the activities of the different types of users, the DBMS can more clearly and accurately delineate the functions available to a user [9.10,

9.14]. For example a personnel department worker can be assigned the corresponding role as opposed to listing each privilege individually. Since the reduced amount of authorized privilege combinations are represented by roles, the unauthorized addition of privileges to a user account becomes more noticeable. Furthermore, the use of roles allows a database administrator to adjust multiple user privileges by adjusting a single role. Figure 9.4, below, shows a role based representation of the information in Figure 9.3.

ROLES			
	Update Records	Remove Employee	Add Employee
Personnel	✓	✓	✓
Account Maintenance	✓	✓	
Benefits	✓		

PRIVILEGES			
	Personnel	Account Maintenance	Benefits
Bob	✓		
Lisa			✓
Mark		✓	
Sarah			✓
Cindy		✓	

Figure 9.4. An example of role based access control.

9.2.4 Trojan Horses

A Trojan horse is a program that performs a task that a user does not expect and does not want completed [9.20]. The name, Trojan horse, stems from the concept that the real functionality of the program is hidden from a user, only rarely making its true purpose apparent after its execution. A detailed discussion of Trojan horses in general can be found in chapter 15. In terms of DBMSs, Trojan horses either modify the database, modify security labels, or modify user roles. For example, a user may execute a DBMS command that creates a backup of a confidential database, that has been altered to make a second copy available for every user to examine. In this example, the backup process functions properly, but the user is unaware of the activity

of the Trojan horse. While Trojan horse programs may go unnoticed to the casual DBMS user, they are preventable [9.4].

The nature of Trojan horse programs lends itself to prevention and possibly detection. Because their presence in a DBMS is an abnormality, they can often be found. This is accomplished by searching for the unusual activity and side effects of a Trojan horse. Searching transaction logs for unusual activity or searching for unexplainable function modifications will often identify the presence of a Trojan horse. This may not be enough, however, as they may have performed their intended task before they are noticed. Nonetheless, a system should take a few preventative measures. These measures are (1) restrict users from introducing additional functionality to the system and (2) frequent examination of existing system operations [9.13, 9.20]. Prohibiting users from adding functionality to the system will prevent the introduction of a Trojan horse. In the event that users cannot or are not prevented from modifying the DBMS, frequent examination of the database operations may locate modified and unauthorized operations.

9.3 Summary

Security in a database management system (DBMS) refers to the promotion of data and operation availability, integrity, and privacy. These concepts are threatened by aggregation vulnerabilities, inference vulnerabilities, and Trojan horses. Both aggregation and inference vulnerabilities allow a user, either authorized or unauthorized, to learn information that is otherwise unavailable to them. Aggregation is accomplished by combining select information from multiple data tables to identify information that is not otherwise accessible. Inference vulnerabilities allow an intruder to deduce the contents of a database without performing any overtly dangerous or suspicious behavior. Unlike aggregation and inference, Trojan horses attack the DBMS operations as well as the data.

Prevention of these vulnerabilities begins with flow controls, inference controls, and basic system security mechanisms. Flow controls, such as multilevel security mechanisms, prevent unauthorized users from accessing or modifying information that an administrator has previously identified as inaccessible. Inference controls, such as data table reorganization, can reduce the possibility of inference. This process is often difficult, however, as every possible inference is not necessarily removed, let alone detected. Basic system security mechanisms, are most useful in preventing Trojan horse attacks. These mechanisms are often similar to those employed by non-database specific operating systems.

9.4 Projects

- 9.1 When does the data aggregation problem coincide with the data inference problem? How can such a situation be prevented?
- 9.2 Why is it important to understand the purpose of a specific DBMS when implementing security in the system? What is the value of knowing which information within a DBMS should be visible to a user and which should be hidden?
- 9.3 Some researchers suggest encrypting the data in a DBMS to prevent aggregation, inference and Trojan horse attacks. Can encryption prevent these types of attacks? If so, how?
- 9.4 Some Trojan horse programs attempt to bypass the DBMS operations by performing their tasks at an operating system level. Other Trojan horses, however, attempt to modify database information through a DBMS's native operations. What effect will a multilevel security system have on this type of attack?

9.5 References

- 9.1 Bell, D. E. and LaPadula, L. J., "Secure Computer Systems: Unified Exposition and MULTICS Interpretation", *Mitre Technical Report MTR-2997*, The Mitre Corporation, Bedford, Massachusetts, April 1974.
- 9.2 Biba, K. J., "Integrity Considerations for Secure Computer Systems, ESD-TR-76-372", *USAF Electronic Systems Division*, United State Air Force, Bedford, Massachusetts, March 1976.
- 9.3 Buckowski, L. J., "Database Inference Controller", Database Security III, D. L. Spooner and C. Landwehr editors, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1989, pp. 311-322.
- 9.4 Cohen, Fred, "Computational Aspects of Computer Viruses", Rouge Programs: Viruses, Worms, and Trojan Horses, Lance J. Hoffman editor, Van Nostrand Reinhold, New York, New York, 1990, pp. 324-355.

- 9.5 Date, C. J., An Introduction to Database Systems, fourth edition, Addison-Wesley, Reading, Massachusetts, Vol. 1, 1986.
- 9.6 Ferraiolo, David and Kuhn, Richard, "Role-Based Access Controls", *15th Annual National Computer Security Conference*, Baltimore, Maryland, October, 1992, pp. 554-563.
- 9.7 Fite, Philip and Kratz, Martin P. J., Information Systems Security: A Practitioner's Reference, Van Nostrand Reinhold, New York, New York, 1993.
- 9.8 Fugini, M., "Secure Database Development Methodologies", Database Security: Status and Prospects, C. E. Landwehr editor, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1988, pp. 103-130.
- 9.9 Hinke, Thomas H. and Schaefer, M., "Secure Data Management System", *Rome Laboratories Technical Report RADC-TR-266*, Rome Air Development Center, Griffiss AFB, New York, November 1975.
- 9.10 Lochovsky, F. H. and Woo, C. C., "Role-Based Security in Database Management Systems", Database Security: Status and Prospects, C. E. Landwehr editor North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1988, pp. 209-222.
- 9.11 Lunt, Teresa F., "A summary of the RADC Database Security Workshop", *Proceedings of the 11th National Computer Security Conference*, Gaithersburg, Maryland, October 1988, pp. 188-193.
- 9.12 Meadows, C. and Jajodia, S., "Integrity Versus Security in Multilevel Secure Databases", Database Security: Status and Prospects, C. E. Landwehr editor, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1988, pp. 89-102.
- 9.13 Reymont Reports, "Detecting and Preventing Misuse of Data processing Systems", Reymont Associates, New York, New York, 1978.
- 9.14 Sandhu, Ravi and Feinstein, Hal, "A Three Tier Architecture for Role-Based Access Control", *17th Annual National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 34-46.

- 9.15 Schell, Roger and Denning, Dorothy, "Integrity in Trusted Database Systems", *9th Annual National Computer Security Conference*, Gaithersburg, Maryland, September 1986, pp. 30-36.
- 9.16 Smith, Gary W., "Solving Multileveled Database Security Problems: Teaching is not Enough", Database Security III, D. L. Spooner and C. Landwehr editors, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1989, pp. 115-125.
- 9.17 Smith-Thomas, Barbara; Chao-Yeuh, Wang and Yung-Sheng, Wu, "Implementing Role Based, Clark-Wilson Enforcement Rules in a B1 On-Line Transaction Processing System", *17th Annual National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 56-65.
- 9.18 Su, Tzong-An and Osoyoglu, G., "Multivalued Dependency Inferences in Multilevel Relational Database Systems", Database Security III, D. L. Spooner and C. Landwehr editors, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1989, pp. 293-300.
- 9.19 Ting, T. C., "A User-Role Based Data Security Approach", Database Security: Status and Prospects, C. E. Landwehr editor, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1988, pp. 187-208.
- 9.20 White, Steve R. and Chess, David M., "Coping with Computer and Related Problem", Rogue Programs: Viruses, Worms, and Trojan Horses, Lance J. Hoffman editor, Van Nostrand Reinhold, New York, New York, 1990, pp. 7-28.
- 9.21 Wiseman, S. R., "On the Problem of Security in Data Bases", Database Security III, D. L. Spooner and C. Landwehr editors, North-Holland Publishers, Stockholm, Sweden, Amsterdam, 1989, pp. 301-310.

9.6 Extended Bibliography

- 9.22 Elmasri, Ramez and Navathe, Shamkant B., Fundamentals of Database Systems, Benjamin Cummings Publishing Company, Redwood City, California, 1989.

- 9.23 Henning, Ronda R. and Walker, Swen A., “Data Integrity vs. Data Security: A Workable Compromise”, *Proceedings of the 10th National Computer Security Conference*, Gaithersburg, Maryland, September 1987, pp. 334-339.
- 9.24 O’Conner, James P. and Gray III, James W., “A Distributed Architecture for Multilevel Database Security”, Gaithersburg, Maryland, *Proceedings of the 11th National Computer Security Conference*, October 1988, pp. 179-187.
- 9.25 Shockley, William R. and Warren, Dan F., “Description of Multiple Secure Entity-Relationship DBMS Demonstration”, Gaithersburg, Maryland, *Proceedings of the 11th National Computer Security Conference*, October 1988, pp. 171-178.
- 9.26 Ullman, Jeffrey D., Principles of Database and Knowledge-Base Systems, Computer Science Press, Stanford, California, Vol. 1, 1988.

10

NETWORK SECURITY

Before the invention of computers, information security was for the most part a physical security problem. Information was protected behind locked doors and kept secure in safes and locked file cabinets. The situation didn't change much with the introduction of early computing systems which were also kept secure behind locked doors. With the advent of remote access card readers and terminals, however, security became harder to maintain since every room with one of these devices had to be kept secure. Additional security was added so that only authorized individuals could use the computer. Authentication techniques, such as accounts with userid/password combinations, were used to verify the authenticity of users. Even with authentication techniques, physical security still played a large part in maintaining control of these early computer systems. With the introduction of networks, however, physical security began to play a less important role. Today, with the global nature of the Internet, physical security can only be counted on to protect the computer system from theft. It can't be relied on to protect the information contained on the computer systems connected to a network. Individuals can now 'travel the digital highways', journeying from the United States, to the Far East, Europe, and back to the United States in a matter of a few seconds. None of the machines except the actual terminal (workstation or PC) the individual is sitting in front of can rely on physical security measures to protect it. Authentication and access control techniques are part of the solution but networks must also rely on other techniques to maintain the integrity of the data they contain. It is these other techniques, used to secure computer systems attached to networks, that is the subject of this chapter.

10.1 Network Fundamentals

Before examining how security is applied to networks, a basic understanding of network organization is required. There are a number of reasons computer systems are connected together in a network including resource sharing, communication, reliability, and increased processing power. Communication between computers in a network can be accomplished in a *point-to-point* fashion where each system transmits to another specific system, or in a *broadcast* manner where systems transmit in general to the media accessible to all other systems. There are several ways that the computer systems can be connected in a network. Figure 10.1 depicts two topologies found in broadcast networks. Figure 10.2 depicts two other topologies found in point-to-point networks.

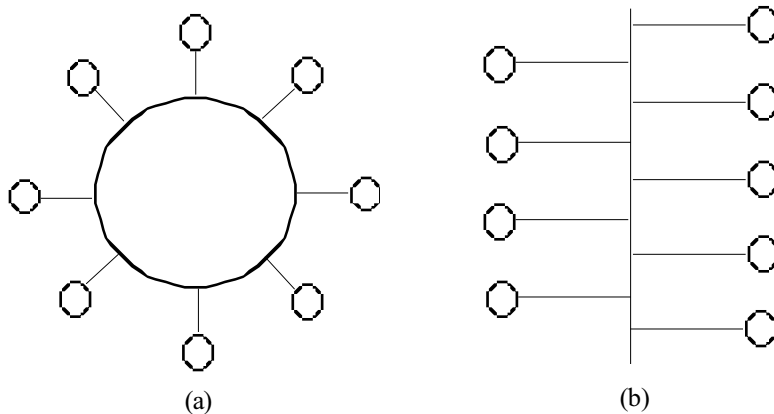


Figure 10.1. Broadcast Network Topologies. (a) Ring. (b) Bus [10.5].

In order to reduce the complexity of networks, most are organized in a series of layers. Each layer is built upon the layer below it to standardize and simplify communication between them. Conceptually, each layer is designed to provide specific services to the layer above it. This effectively hides the details of communication between lower level layers and the upper levels and serves to modularize the protocols for each layer. The purpose is to also create an environment where each layer on a system communicates with the same layer on another system using the protocol developed for that layer. Each subsequent lower level takes the

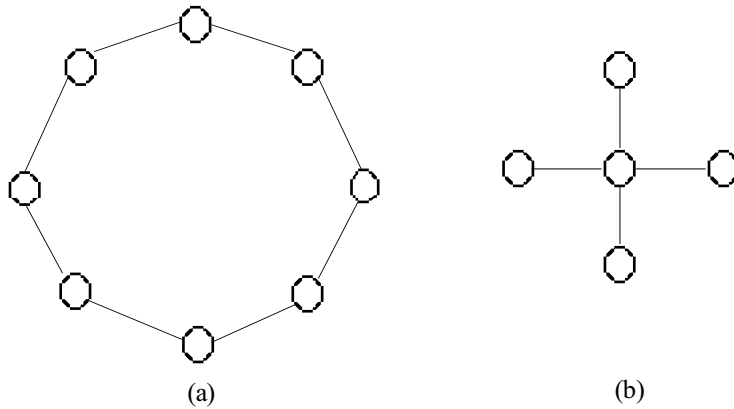


Figure 10.2. Point-to-point Topologies. (a) Ring. (b) Star [10.5].

information passed to it from the level above and formats it to meet its protocol needs before sending it along. [Figure 10.3](#) illustrates this process.

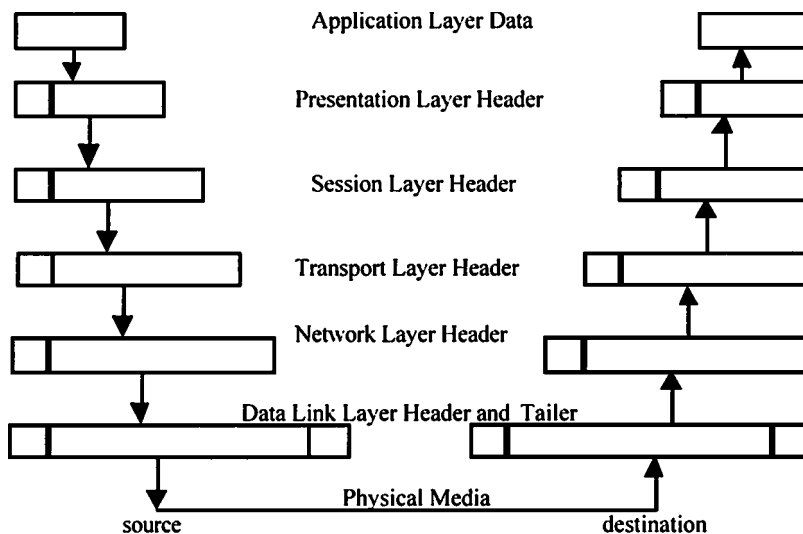


Figure 10.3. Packaging of data in layers [10.3].

In this example, a seven-layer model is assumed. Data to be sent is created at Layer 7 of the source host destined for Layer 7 on the destination host. As far as the application at Layer 7 on the source host is concerned, the message only needs to be formatted in the Layer 7 protocol, and is then passed to the next level below, Layer 6. At this next layer, the message is sent along formatted so that it conforms to the Layer 6 protocol and is passed to Layer 5 where the process continues. It is possible at any of these layers that the message may have to be split into different parts. At any of the layers a header or trailer may also be attached to the message. If the message is split, each portion is treated as its own entity by all layers below the split. Each portion may then receive its own header and/or trailer. The packaged data is eventually passed to the physical media where it is transmitted to the destination source. As the message is received at the Destination Host, it is passed back up through the layers. Each layer strips off its header and if the message had been split into separate parts, reassembles it. The advantage of this layered approach can be seen at Layer 3 and below where splitting of packages normally occurs. The application program at Layer 7 does not have to worry about splitting a large message into parts, this will be handled at the lower layers where constraints are imposed by network protocols. The lower layers in turn don't worry about the contents of the message, this is the responsibility of the application layer.

The standard model of a layered network architecture is the 7-layer International Standards Organization (ISO) Open Systems Interconnection (OSI) Reference Model. This model, including what is known as the communication subnet, is depicted in [Figure 10.4](#).

The lowest layer of this model is called the **Physical Layer**. This layer is concerned with the actual transmission of raw binary bits across a transmission medium. Error detection and correction is not a concern at this layer. Layer 2, **Data Link**, is designed to take the raw bits that have been transmitted and to turn them into what appears to be an error-free line. The next layer up from Data Link Layer is the **Network Layer**. This layer is concerned with the controlling of the subnet. A key issue at this layer is routing. The **Transport Layer** is the fourth layer whose purpose is to provide transmission services to the higher levels without being concerned about cost-effective data transfer. It also insures that all pieces are received correctly. Layer 5 is the **Session Layer** which provides a way for higher level entities to establish synchronized dialogue (sessions). The **Presentation Layer**, layer 6, provides certain services that are frequently used by the seventh layer. An example of this is data compression. The highest level, layer 7, is the **Application Layer** which is concerned with a number of different protocols. An example of this is the communication necessary to invoke a full screen editor across a network.

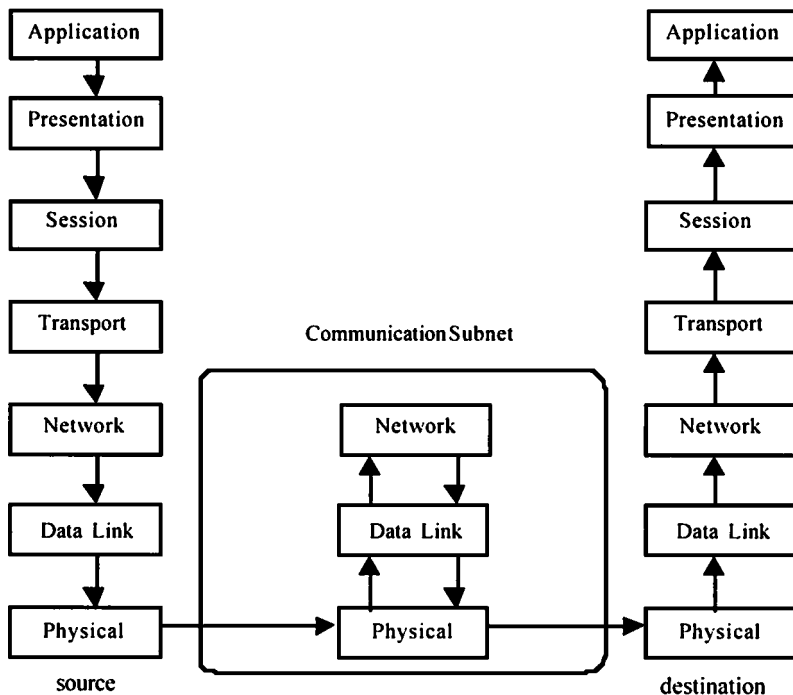


Figure 10.4. OSI Reference Model [10.5].

The communication subnet depicted in [Figure 10.4](#) is designed to transmit data between different hosts without the user having to be concerned with routing issues. Often the machine the user is connected to is not the machine the user ultimately wants to communicate with. Any data transmitted between the two will need to be passed to several intermediary machines to relay the information. Several different networks may in fact be involved with the transmission. The entire seven layers worth of protocol packaging does not need to be unpackaged in order to transfer the data, only the subnet portion needs to be addressed.

In practice, the entire OSI model is not implemented. The most common layered set of protocols in use is the Transmission Control Protocol/Internet Protocol (TCP/IP) suite of protocols. TCP/IP works in a very similar manner to the OSI model in that it takes a layered approach to providing network services. Each layer in the TCP/IP model communicates with the layers above and below it in the same way that

the layers in the OSI model do. The TCP/IP protocol suite consists of five layers roughly equivalent to the OSI model. The specific layers implemented are the application, transport, network, data link, and physical layers. The lower layers serve the same function as previously described for the OSI model. The application layer in the TCP/IP suite provides the same services as do the top three layers of the OSI model.

Each of the top three layers of the TCP/IP protocol suite actually consists of multiple protocols. At the transport layer, for example, the two common protocols are the Transmission Control Protocol and the User Datagram Protocol (UDP). In the application layer can be found protocols such as the File Transfer Protocol (FTP) and the Simple Mail Transfer Protocol (SMTP). It is not uncommon to hear these protocols mentioned in discussions of security as problems in the protocols are often exploited by individuals attempting to gain unauthorized access to computer systems and networks.

Another issue related to the services provided by the various layers is whether the service provided is *connection-oriented* or *connectionless*. A connection-oriented service is one in which the connection between the same layer on two different hosts consists of several phases; the establishment, transfer and termination phases. The transfer phase is where the actual stream of data units is conveyed between the two. The data is generally guaranteed to be received in the proper order and without transmission errors. Connectionless service, on the other hand, involves the transmission of single data units without any guarantee that the units will be received correctly or in the proper order. In addition, it is also possible to receive multiple copies of the data units at the destination host. TCP is a connection-oriented protocol while UDP is a connectionless protocol.

10.2 Network Security Issues

Network security isn't any different than single host security in terms of its goal. Network security is still aimed at meeting the objectives of providing confidentiality, integrity, availability, and access for legitimate users of the network resources. The real difference in providing basic security services occurs because of the increased complexity of the networked environment. Providing confidentiality of information, for example, is difficult enough when the entire system resides in a single room. Consider the implications of allowing access to information from multiple locations located in areas separated by tremendous distances. Security for a single host is generally the responsibility of a single individual. In a networked environment, the security of individual systems is the responsibility of numerous individuals. Intruders

to networks continually count on finding a single weak link in the network chain which will then allow them access to the rest of the network. Network security measures must account for this, as well as other complexities, in an attempt to maintain the security of the network data and resources.

10.2.1 Basic Network Security Objectives and Threats

The security services of a network have four fundamental objectives designed to protect the data and the network's resources. These objectives are:

- **Confidentiality**: Ensuring that an unauthorized individual does not gain access to data contained on a resource of the network.
- **Availability**: Ensuring that authorized users are not unduly denied access or use of any network access for which they are normally allowed.
- **Integrity**: Ensuring that data is not altered by unauthorized individuals. Related to this is **authenticity** which is concerned with the unauthorized creation of data.
- **Usage**: Ensuring that the resources of the network are reserved for use only by authorized users in appropriate manner.

Opposing these objectives and the network security services are a number of threats. These threats can be described in terms of how they affect the normal flow of information in the network. There are four basic patterns of attack for these threats as depicted in [Figure 10.5](#). The first of these is **denial of service** in which the flow of information is blocked entirely. This can be accomplished in a number of ways including affecting the medium through which that data must travel or the source host itself where the data (or requested network service) resides. The second pattern of attack is **modification** where the contents of messages or the data itself is modified before it is received at the destination host. The third pattern of attack is **interception**. In this attack the normal flow of information is not affected, instead an additional flow, generally to an unauthorized source, is created. Two examples of this form of attack are **eavesdropping**, where another (unauthorized) user gains access to the information as it is transmitted, and **traffic analysis** where information about the network, its services, and its users is obtained by observing the type, destination, and volume of traffic without knowing the contents of the messages or data sent. The last attack pattern is **creation** in which new data traffic is created and inserted onto the network, generally masquerading as data from another, authorized, source.

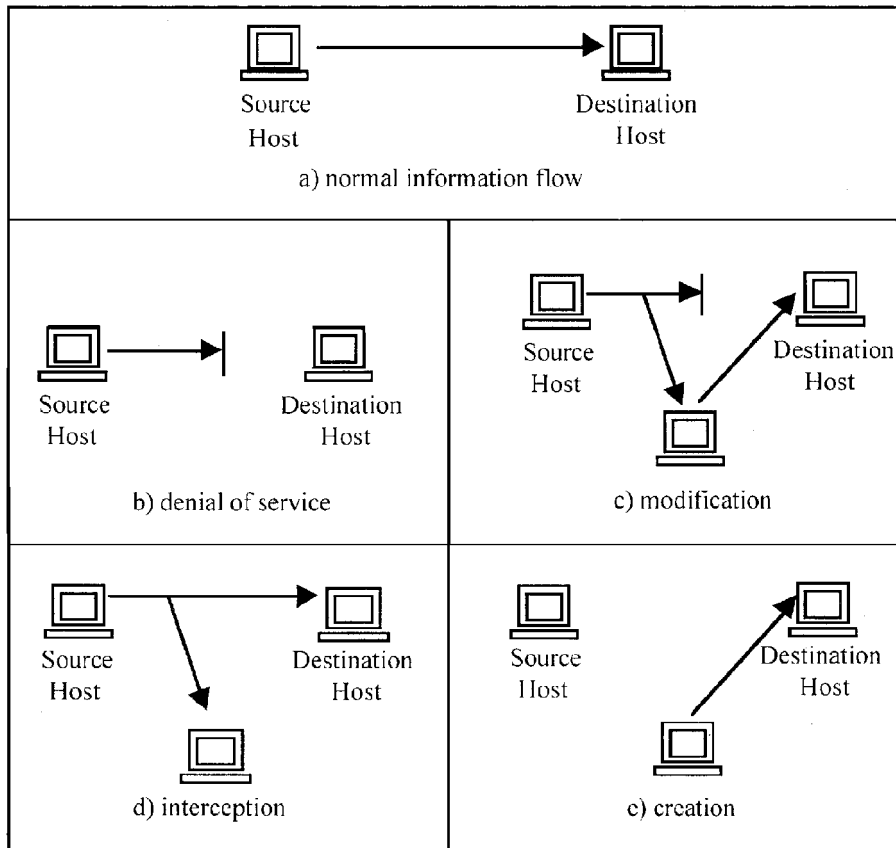


Figure 10.5. Patterns of network attacks [10.4].

10.2.2 Security Services

Defending the network from these patterns of attack are a series of six *security services*. These services are:

- **Access Control:** Protects against unauthorized use of the network or its resources.
- **Authentication:** Establishes the identity of an entity (either a user or another system). Also prevents repudiation of an exchange of previous traffic by either the sender or receiver.

- **Confidentiality** Protects information from being disclosed to unauthorized entities.
- **Data Integrity**: Prevents the unauthorized modification or deletion of data.
- **Traffic Flow Integrity**: Prevents the collection of information about the network through observation of network traffic characteristics.
- **Assured Usage**: Prevents the denial of service through degradation of network services.

It is important to be able to visualize the environment for these services if we are to understand the difficulty of implementing them. In [Figure 10.6](#) a connection between two hosts has been requested. This connection can be for any of a variety of reasons such as sending or receiving files, electronic mail, or for requesting a specific service. The local networks each host is part of are not directly connected to each other so the transmission between the two must travel through several other intermediate subnets. The path the transmissions take may include different configurations of topology and may include networks supporting different services. In theory the subnets should simply determine that the transmission is not intended for them and forward it onward. In practice, however, the two end systems can not be assured that the subnets simply forward the data and thus must rely on the network security services to provide the necessary assurances.

The authentication service insures that the source for the received data is correctly identified. Related to authentication is **non-repudiation**. Often it is important to be able to prevent a sender of a message from denying that it was sent. This is especially true in financial transactions. Similarly, it is also often important to be able to determine that the destination host did indeed receive the message. Another problem that must be addressed by both the authentication and access control servers is the replaying of previously captured transmissions. This technique has been used by unauthorized individuals to trick hosts into believing that the transmission came from an authorized individual. In [Figure 10.6](#), for example, a user on one of the subnets not normally authorized access to the destination host could capture login information which had been transmitted at an earlier time from an authorized user on the source host to the destination host. This information could later be replayed to the destination host in order to obtain access to it. Finally, the authentication service is often used in conjunction with other mechanisms to provide the access control service.

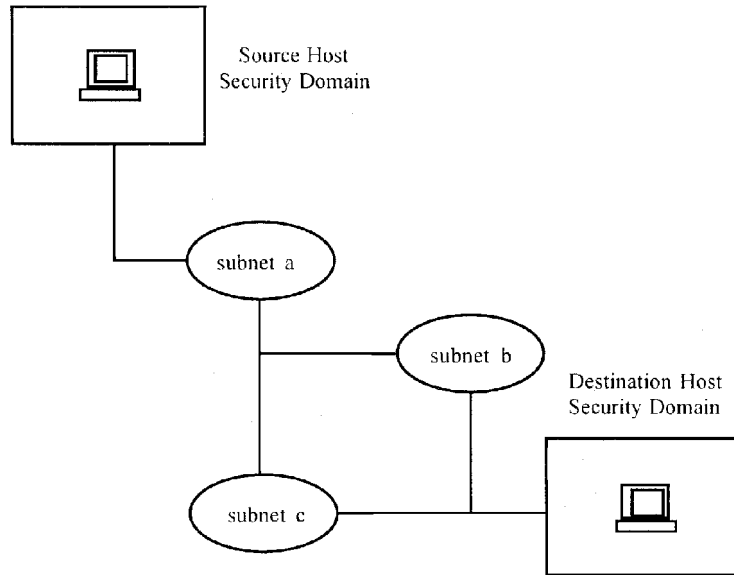


Figure 10.6. Transmission between networks.

Access control is concerned with protecting against the unauthorized use of the network or its resources. This service directly contributes to the objectives of confidentiality, availability, and integrity by enforcing authorization. Authorization generally assumes a pair of active entities; the *target* and the *initiator*. Authorization, and thus access controls, govern which initiators may access and use which target systems or network services. Access control is also responsible for a process called *selective routing*. The purpose of selective routing is to prevent sensitive information from being transmitted through networks or subnets which are not trusted. In Figure 10.6, for example, if subnet b was not trusted by the source host but subnet c was, selective routing would be responsible for insuring that important transmissions from the source to the destination hosts would avoid subnet b.

Confidentiality services protect transmissions from being revealed to unauthorized users. There are two basic techniques used to provide this service. The first is to trust only entities in a well defined *security domain*. A security domain consists of all hosts and resources, as well as the transmission medium used to connect them, which abide by a formal security policy and for which we can be assured of a certain level of security. Hosts in this domain place a certain level of trust in the other hosts and may thus provide certain services for these trusted hosts which are not available to hosts

residing outside of the security domain. An example is the **UNIXrlogin** command which allows a user on one host access to another host without having to provide a password for this new host. The security domain may also consist of additional networks or subnets for which the same level of trust has been placed. The second technique is to hide important information through the use of **encryption**. Encryption transforms the information to be hidden from an understandable format to one which is unintelligible to any but the intended recipient (who knows how to transform the information back to its original format). This technique is used whenever a transmission has to leave the current security domain in order to travel to its ultimate destination. Since we can't trust systems that are outside of our security domain, we must hide the important information from those who might try to intercept it.

Encryption is also useful in another security service, traffic flow integrity. In this service the goal is to prevent unauthorized individuals from obtaining information through observing characteristics of the transmission itself. Characteristics of importance include the source and destination, frequency, and size of the transmission. The difference between encryption for traffic flow purposes and encryption for confidentiality is the level of the OSI model the encryption is performed at. If, for example, an application running at the seventh layer of the model is communicating with another application on a different host, the data can be passed to the sixth layer and encrypted at that point before passing it along. The information is then sent through the network and isn't decrypted until it arrives at the destination host whose sixth layer knows how to decrypt this specific data. If the data is captured in any subnet between the source and destination hosts the information will be useless since it can not be understood. Encryption at this level is known as **end-to-end** encryption since the encryption process takes place only at the source and destination hosts (the two ends).

Encryption of only the data does not help with traffic flow integrity since the other characteristics of the transmission can still be observed. An alternative approach is to encrypt as the transmission leaves the host. This will encrypt the entire transmission which will also serve to hide the transmission characteristics. Of course the transmission will need to be decrypted at each node in the network or subnets so that it may be properly routed, but it does protect the transmission from analysis through wiretapping or electronic emissions from the physical medium. This form of encryption is called **link** encryption since it is performed at each link of the transmission chain between two communicating hosts. If one can trust all intermediate systems and subnets, this effectively provides the traffic flow integrity desired. A final point on traffic flow analysis; it should be obvious that more than just encryption is needed if one wants to hide the volume of traffic sent from or to a specific host or network. For this reason some organizations (the military in particular) have systems generate false messages during periods of low network usage. These messages are

simply ignored by the receiving host but will appear to be just as real to anybody who has tapped into a transmission line as other, official traffic.

The data integrity/security service ensures that data is not modified or deleted by an authorized individual. In terms of a network, the concern about modification is not only at the site where the data is stored but also as it is transmitted. It also includes the unauthorized creation of new data items supposedly from an authorized source. While both data integrity and confidentiality services share similar methods to provide their respective service, it is important to understand the difference between them. Suppose, for example, the source and destination hosts in [Figure 10.6](#) represented a bank and one of its remote automated teller machines (ATM). An individual wishing to exploit the system might attempt to tap into the line between the two systems so that a customer's account and personal identification number (PIN) could be obtained. This would be a confidentiality issue. If, on the other hand, a customer wishing to exploit the system attempted to modify the transmission between the two systems (so that the ATM dispensed \$100.00 but the bank only received a request for and authorized a \$10.00 transaction) it would be a data integrity issue.

Encryption will help with the modification portion of data integrity since the customer would not know what to change in the encrypted message. It will not by itself, however, solve the associated problem of replay. Suppose, for example, that a customer were able to capture all of the transmissions from one transaction between the bank and ATM. Later the customer accesses the transmission line and attempts another transaction. This time, however, the customer blocks all messages to the bank from the ATM and simply resends the authorization recorded earlier thus convincing the ATM to dispense the money without the bank receiving notification of the transaction. One way around this problem is to provide a time-stamp inside of the encrypted message. Thus when the recorded transmission is replayed the time-stamp will reveal the fraudulent nature of this transaction. A field or value included for integrity purposes is often referred to as an integrity check value.

Finally, the assured usage service is concerned with the availability of network resources to authorized users based on some defined level of performance. This defined level of performance is often referred to as the Quality of Service (QoS) and should be stated in the security policy. This service is often more difficult to ensure than the others. It may also actually result in competition between authorized users since the security policy may define different levels of users and may sacrifice service to one level in order to provide service to another level.

10.3 The Trusted Network Interpretation

The Department of Defense has provided guidance as to what is required to secure a network. This guidance is outlined in the *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* [10.2]. Commonly referred to as the *Red Book* or the TNI, this guidance provides an interpretation of how the *Orange Book* can be applied to a network environment. The first part of the *Red Book* is the actual interpretation of the various classes found in the *Orange Book* and how they apply to networks. A second part of the *Red Book*, however, is a list of other security services that either do not exist on a stand-alone system or that have increased significance in a network.

There are two viewpoints from which to evaluate secure networks. The first is the Interconnected Accredited Automated Information System View. This viewpoint looks at a network as a series of multiple interconnected Automated Information Systems (AIS) that have been individually accredited to handle certain levels of information. Each AIS will agree with another AIS as to what range of sensitive information can be exchanged between them. A network designed this way really consists of a series of systems, each of which is responsible for enforcing its own security.

The second viewpoint is the Single Trusted System View in which a common level of trust is supported throughout the entire network. A single trusted system is viewed as a single entity rather than a series of individual components. The network formed adheres to a single security policy and is considered a single trusted computing base, referred to as the Network Trusted Computing Base (NTCB). The functions of the NTCB may be partitioned among the various systems connected to the network but all work together to ensure that the overall security policy is enforced.

In terms of the network interpretation of the *Orange Book*, it is generally not practical to apply the various requirements to a network consisting of interconnected AIS. Instead, Part I of the TNI is aimed at networks with a single NTCB viewpoint. Part II, however, is applicable to all networks, no matter what structure they take. We will not discuss the individual network interpretation of any of the various *Orange Book* requirements but will instead concentrate on the security services outlined in Part II of the TNI.

10.3.1 TNI Security Service

The TNI addresses nine security services split evenly among three categories. The categories are *Communications Integrity*, *Denial of Service*, and *Compromise Protection*. For each service, the TNI lists both a brief functional

description of the service as well as possible techniques to provide the service. It also lists three criterion (Functionality, Strength, and Assurance) for each and the range of possible evaluations. While the criterion for rating the services is not important in a general discussion of network security, the description of the services themselves, as well as examples of possible mechanisms for implementation are and provide a glimpse of other factors that need to be considered in securing a network.

The first category, ***Communications Integrity***, is concerned with the accuracy, non-corruptibility, and level of confidence in the transmission between two systems in the network. This first category consists of the three services ***Authentication, Communications Field Integrity***, and ***Non-repudiation***.

Authentication is concerned with ensuring that an established transmission takes place between the desired entities and not an entity attempting to masquerade as another or to replay a previously captured transmission. If this service is provided for a connection-oriented transmission the TNI refers to it as Peer Entity Authentication. If it is in support of a connectionless transmission it is called Data Origin Authentication. The methods listed to implement this service include encryption, something known by the entity (e.g., a password), and use of some characteristic of the entity (e.g., biometrics) or a possession owned by the entity (e.g., smartcards).

Communications Field Integrity refers to the protection from unauthorized alteration of the fields in the data transmitted between two entities. This includes any of the header, trailer, and data fields. The TNI does allow the network architect, administrator, and even the user to selectively apply data integrity to certain fields depending on the ultimate security goal of the designer. Countermeasures for this service include policy, procedures, and automated or physical controls. Encryption, however, is generally the primary technique used.

The non-repudiation service ensures that there is an unforgeable proof of both shipment and receipt of data. This means that neither the sender nor receiver can deny that a legitimate message was sent and received. One technique to implement this service is called Digital Signatures (discussed in chapter 14) which employ two parts; the signing of a data unit and the verification of a signed data unit. The signing process is accomplished by either enciphering the data unit itself or producing a special cryptographic checksum of the data unit. The verification process employs the appropriate decryption process to determine whether the signature was produced by the stated signer.

The second category of service described by the TNI is ***Denial of Service***. This entails maintaining the network at some predefined level of throughput and also maintaining the availability of any remote entity to authorized users. The TNI also states that a denial of service (DOS) condition exists whenever network resources are not available to users on an equitable basis although priorities can be used in the

determination of equity. During active times on the network, a DOS condition can be detected by not receiving a response for a specified maximum waiting time. During periods of inactivity, however, DOS conditions are harder to detect unless periodic messages are sent to test connections. It is important to note that DOS conditions may occur either as a result of an attack by an outsider designed to deny the use of the network to authorized users, or simply through failure of the network to handle an extreme or unanticipated load. The three specific services contained in the DOS category are ***Continuity of Operation, Protocol Based DOS Protection Mechanisms***, and ***Network Management***.

Continuity of Operations, as its name implies, refers to the assurance that the network is available for use by authorized individuals. This assurance is based on a pre-specified minimum level of service. Approaches to handle DOS conditions include the use of redundant components, reconfiguration of existing network assets, fault tolerance mechanisms, and the distribution of network control functions among the various network systems. An important aspect of this service takes place when the network is implemented and tested. This testing should include service response level measurement under expected extreme conditions to determine the network's ability to handle extreme traffic loads..

Protocol Based DOS Protection Mechanisms include those measures which use existing or additional protocol mechanisms to detect DOS conditions. Where possible, existing protocols should be used in order to limit network traffic overhead which may by itself lead to a DOS condition. A common protocol defined to detect DOS conditions is the request-response mechanism which uses periodic "are-you-there" messages to determine the existence of an open path between entities. Another method to detect DOS conditions might involve the measurement of throughput between two entities which use input queuing. Should the measured throughput fall below a predefined minimum a DOS condition exists.

The TNI Network Management service describes those measures taken to configure the network or monitor its performance which aren't described by current protocol models. Using throughput as a measure of performance may not be sufficient, for example, since service could drop below defined performance standards due to an extremely noisy channel condition. As a result, the TNI takes a two tier approach to DOS resistance. The first tier was the protocol based mechanisms. The Network Management service provides the second tier. This second tier deals with the health of the network, detection of failures, and identifying overt acts which degrade the service of the network. The second tier is not concerned with lower tier measurements such as throughput.

The last category of service defined by the TNI is ***Compromise Detection***. This category is concerned with the non-disclosure of information that is transmitted

between two entities of the network. The three specific services covered by this category are *Data Confidentiality*, *Traffic Flow Confidentiality*, and *Selective Routing*.

The data confidentiality service protects data from unauthorized access. The main method used to obtain unauthorized access to data is passive wiretapping which is aimed at the data itself and not at the other fields used in transmission. Consequently, this service is mainly aimed at protecting against this form of attack. The two principle methods used to implement this service are physical security (keep potential wiretappers physically away from the systems and transmission medium) and cryptography.

Traffic flow confidentiality services are concerned with the fields other than the data field used in a transmission between two entities, as well as other characteristics of the transmitted data itself. This includes the size of the message, the frequency of message traffic, and the source and destination of each transmission. While encryption above the transport layer will effectively hide the data, it does not protect against information obtained from these other sources. The three methods used by the traffic flow confidentiality services to perform their function are physical security (if an individual can't get close enough to the systems or medium to perform a passive wiretap they can't obtain any information), encryption (below the transport layer), and traffic padding. The problem with encryption applied below the transport layer is that, while it protects against an analysis of fields relating to such things as the source and destination, it also requires all intermediate systems in the transmission path be secure. This is necessary since, at each intermediate system, the transmission data must be decrypted in order to send it to its ultimate destination. The purpose of traffic padding is to maintain a certain transmission level so that, whether during periods of little network activity or heavy usage, the flow of data appears to be constant. For this technique to be effective, the spurious transmissions need to be encrypted so they are not detected as spurious but appear to be valid data.

The final security service, selective routing, is designed to control the path transmitted data takes from source to destination. This control allows the data to either choose or avoid specific networks or links. This means that, if required or desired, only physically secure subnets could be used to transmit the data concerned, thus lessening the risk during transmission. This service can be implemented to allow any combination of either the initiator of a transmission selecting the route (or specific subnets or systems to avoid), or the end-systems dynamically changing routes based on persistent evidence of attacks.

10.3.2 AIS Interconnection Issues

There are two views that can be taken when discussing interconnection of Automated Information Systems. The first is the **Component Connection View**. In this approach, any AIS which is connected to another AIS must enforce an **interconnection rule** which specifies what level of information may be transmitted or received. Each system is then responsible for maintaining separation of multiple security levels of information and determining what information can be sent or received. This means that each system does not need to know the range of possible security levels for all other systems, just the range of levels for its immediate neighbors. The **Global Connection View**, on the other hand, requires that each AIS know the range of security levels processed for all components in the network. While the component connection view is somewhat simpler to implement, it is easier to spot potential problem, such as the **Cascading Problem**, when the network is addressed from a global connection standpoint.

The cascading problem is a description of a condition that can exist when various systems with different security levels are interconnected. [Figure 10.7](#) illustrates the problem associated with this type of connection. In this figure there are two systems, each designed to maintain separation between two security levels. The first maintains separation between Top Secret (TS) information and Secret (S) information. The second maintains separation between Secret and Unclassified (U) information. Neither

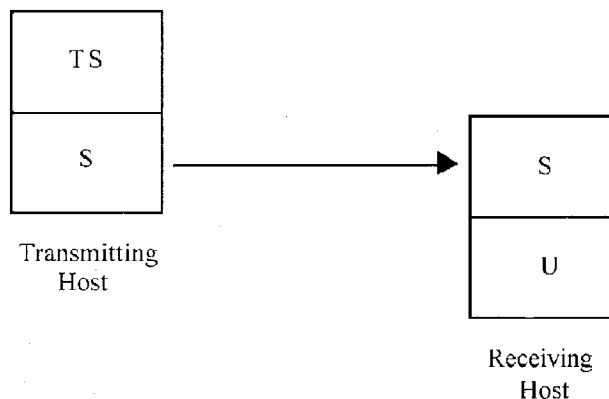


Figure 10.7. Cascading Problem [10.2].

system has the safeguards in place to guarantee separation between three security levels. If the Transmitting Host fails to maintain its separation of data and Top Secret information is allowed to be accessed by a user cleared for only Secret, this data could be sent to the Receiving Host. If the Receiving Host in turn fails to maintain its separation of information this Top Secret information may be accessible (maybe through the same technique as in the Transmitting Host) by a user only cleared for Unclassified information. In essence, the network is processing information requiring three security levels but its individual systems are only trusted to maintain separation between two levels. The level of trust required to maintain separation between three levels is much greater and would require enhanced security mechanisms. The problem, as illustrated in [Figure 10.7](#), can be detected from both the Global and Component Connection viewpoints. But consider the extension to the network as illustrated in [Figure 10.8](#).

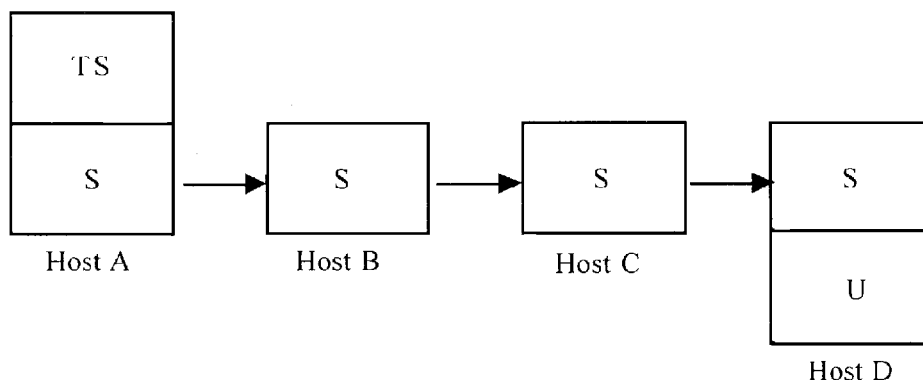


Figure 10.8. Extended Cascading Problem.

In this second example of the cascading problem, the transmitting host, Host A, is connected to an AIS, Host B, which only processes information at one level. Should Host A's safeguards to separate the Top Secret information it processes from Secret only users fail, the level of compromise is no greater should the data subsequently find its way to Host B. The same is true if this information is further sent on to Host C which is also cleared for only Secret users. Host C, however, is connected to another AIS which is processing information at two security levels and if the Top Secret data, now labeled as Secret, finds its way to Host D, whose separation safeguards also fail, the Top Secret information could be obtained by users only cleared for Unclassified. Looking at the network from a Component Connection viewpoint, where only the

immediate neighbors are of concern, the potential for this problem would not be spotted. It is only from the global viewpoint that this possibility would be spotted. This illustrates the need to take a global point of view when making a decision on whether to grant access to additional hosts in a network.

Another AIS interconnection concern is what is referred to as the ***Propagation of Local Risk***. This occurs when the administrators for a host have decided to accept certain safeguards which might normally be considered adequate for the information or levels of information being processed. Often such a decision is based on operational needs for the organization involved. If this AIS is connected to a network, however, other systems connected to the net need to know of the acceptance of risk by the administrators of this AIS so that appropriate decisions concerning data transmissions and receivals can be made. Otherwise, the same risk accepted by the one AIS will be passed along to the other systems connected to the network.

10.4 Distributed Systems Security

A recent trend in computing has been towards what is referred to as distributed systems. A distributed system is a combination of hardware and software components connected via a network. The difference between a distributed system environment and a traditional network, however, is that in a distributed environment the software components may execute on two or more of the hardware processors which communicate via the network. The reasons for implementing a distributed system are similar to those for a network, namely resource sharing, reliability, and communication. Added to this list is speed, since in the distributed system a problem may be divided amongst several processors which together may be able to solve the problem faster than if a single processor had been assigned the task. A common technique seen in distributed computing is to assign certain services to specific processors. A user needing to access this service will need to communicate with the processor assigned this task. The process of accessing this remote service is done transparently to the user. From the users perspective, all services appear to reside on the systems they are directly connected to.

With the increase in capability provided by distributed systems comes additional security concerns. This is especially true when certain services, such as the security services themselves, are distributed among various processors. Consider, for example, the problems associated with remote access of security services. In order for a user to demonstrate appropriate credentials to all other servers when desiring to obtain access to their services, the user's authority to use the service must be validated. Since the authentication server resides on a separate processor, a method is required to demonstrate

to the requested server that the user is authorized access. This is often done through the use of a *ticket* which has been granted by the security server. This ticket serves as a pass which the user gives to the processor running the desired service to validate the user's authority. The ticket is generally encrypted using data known to the desired server and the security server but not to the user. In order to avoid later replay problems, the ticket may also contain a time-stamp or similar technique. This process is illustrated in [Figure 10.9](#). In this example, the user first requests the ticket for the desired service from the security server. The server verifies the user's authority for the desired service and then responds with the encrypted ticket. This ticket is then sent to the desired server which decrypts it and verifies the user's authority. The user is then allowed access to the service.

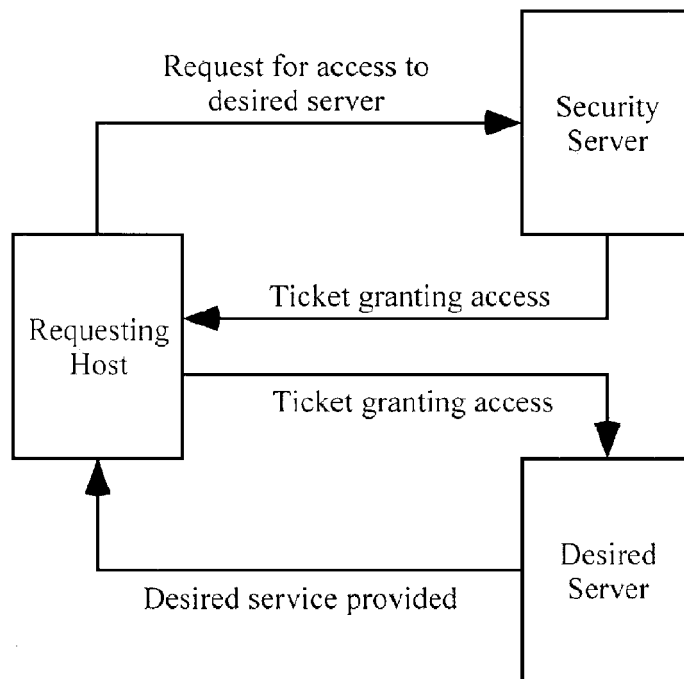


Figure 10.9. Authenticating Access to Distributed Services.

Another problem associated with distributed systems is *delegation of authority*. This occurs when the user requests service from a server which, in order to fulfill the request, must access another server. This problem is illustrated in [Figure 10.10](#). In this illustration, the user has requested a service from Server A. This server

in turn requests Server B perform some operation in order to satisfy the user's request. The problem is how to prove to Server B that the user is authorized access to its services. One possible way to solve this problem is to have the user provide tickets for all possible target servers but this would require that users know in advance all the servers they might need. Another solution is to expand the ticket granting function in the Security Server to include tickets for all other servers that the requesting service may need to access. Which solution is best, or if neither is acceptable and a hybrid approach utilizing elements of both approaches may be better, depends on the specific distributed environment and the services offered.

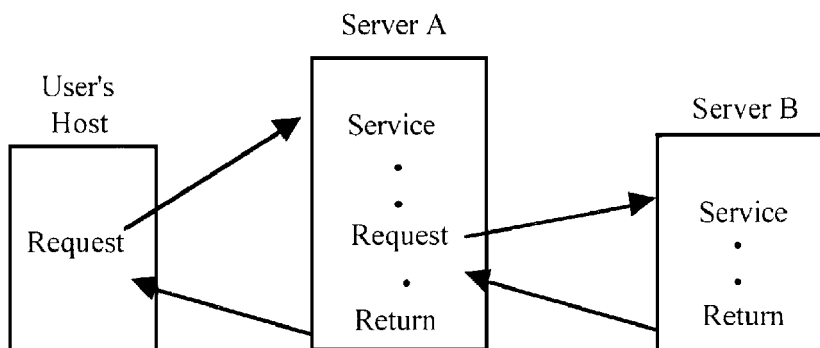


Figure 10.10. Delegation of Authority in Distributed Servers [10.1].

10.5 Modem Access to the Network

It is not uncommon in today's heavily networked environment to have one or more modems attached to systems connected to an organization's TCP/IP network. Often these devices are connected without the knowledge of security administrators. Unauthorized modems attached in this manner are referred to as "rogue modems." When attached to a system inside of an organization's security perimeter they can effectively create a path to circumvent that perimeter.

The common approach to addressing the problem of rogue modems is to use telephone scanners, or war dialers, to probe an organization's telephone network in order to discover the devices. War dialers obtained their name from the 1983 movie *WarGames* in which the star used a computer with an attached modem to scan for telephone lines connected to computers. Since that time, a number of commercial and

freeware war dialers have been created. The problem with telephone scanners is more in the way they are used as opposed to the software itself. Scans are commonly performed at night so as not to disturb the normal operation of the organization being scanned. Scans are also generally performed on a very infrequent basis because they are time consuming and laborious. This results in what can best be described as a snapshot of the network at the time the scan is conducted. A modem attached seconds after the scan is completed will go unnoticed until the next scan is performed. While important, these scans are only of limited value if not performed frequently. War dialers can generally detect the difference between telephone lines being used for data, fax, or voice transmission and some are even capable of performing simple penetration tests of software detected on the lines.

A recent development to address the issue of modem security is the creation of firewalls for the telephone network. The purpose of these devices is very similar to firewalls for the TCP/IP environment – they protect the internal telephone network from the public telephone network by enforcing an organization's security policy for the network. They can also detect the difference between data, fax, and voice transmissions on a line and can terminate calls on lines when the allowable type does not match the current use (e.g., the telephone firewall can block data transmission on a line that has been designated as voice only by the organization's security policy). Further research is being conducted to have these devices also perform intrusion detection on telephone lines in a manner similar to intrusion detection devices in the TCP/IP environment. Key research is also being performed to consolidate all data from the various network sensors (firewalls, routers, intrusion detection systems, and telephone firewalls) at one location so that a comprehensive view of the network and its overall security status can be obtained.

10.6 Summary

The increase in the number of systems connected to networks and distributed systems connected via networks seen in recent years will continue. With networks, an additional level of complexity for security services is introduced. Many of the issues relating to network security are similar to single host environments. Network security is still concerned with the basic security goals of Confidentiality, Availability, Integrity, and Usage. One technique often proposed to solve network security problems is encryption. While encryption does indeed help with certain aspects of the network security problem it does not solve all problems. Additional measures such as traffic padding and selective routing may need to be implemented to provide the level of security required for the information being transmitted. Even if all of these measures

are implemented, the very nature of a network which connects dissimilar systems, often processing information from varying security levels, introduces problems. Often the level of trust one system places in its safeguards may not be shared by other systems on the network. Unless care is used connecting such systems, a propagation of this local risk may ensue. In addition, the network must be viewed from a global viewpoint to avoid the cascading problem where the network as a whole may be processing information at more security levels than any one system can appropriately safeguard.

10.7 Projects

- 10.1 A Token Ring network is a point-to-point network in which the systems are configured in a ring. Use of the medium is governed through the use of a token passed from system to system. A system can not transmit data unless it has the token. If, upon receiving the token, a system has data to transmit, it does so and then sends the token along upon completion. If it has no data to transmit it simply sends the token along. What are two easy methods to affect the performance of this type of network and thus potentially deny the use to authorized users?
- 10.2 Propose a solution to the extended cascading problem described in the text. Will detection or prevention be easier for this type of problem?
- 10.3 Two possible methods to handle the delegation of authority problem in distributed systems were discussed. Both methods have associated problems. Describe the problems associated with each and propose solutions for them.
- 10.4 Covert channels have been described in a previous chapter. Describe possible network covert channels and provide methods to detect and/or prevent them.
- 10.5 A current trend in networking is towards the use of wireless transmissions. How will this affect network security? Be sure to address each of the network patterns of attack and the security services that address them.
- 10.6 What, if any, would the differences be in the security services provided for connection-oriented versus connectionless networks?

10.8 References

- 10.1 Lockhart, Harold W., Jr., OSF DCE: Guide to Developing Distributed Applications, McGraw-Hill, Inc., New York, New York, 1994.
- 10.2 National Computer Security Center, Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria, NCSC-TG-005, Version 1, 31 July 1987.
- 10.3 Pooch, Udo W.; Machuel, Denis and McCahn, John, Telecommunications and Networking, CRC Press, Inc., Boca Raton, Florida, 1991.
- 10.4 Stallings, Willian, Network and Internetwork Security: Principles and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- 10.5 Tanenbaum, Andrew, Computer Networks, 2ed, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

10.9 Extended Bibliography

- 10.6 Arsenault, Alfred, “Developments in Guidance for Trusted Computer Networks”, *Proceedings of the 10th National Computer Security Conference*, September 1987, pp. 1-8.
- 10.7 Branstad, Dennis, “Considerations for Security in the OSI Architecture”, *Proceedings of the 10th National Security Conference*, September 1987, pp. 9-14.
- 10.8 DeMillo, Richard and Merritt, Michael, “Protocols for Data Security”, *Computer*, Vol. 16, No. 2, February 1983, pp. 39-50.
- 10.9 Donaldson, Albert; McHugh, John; and Nyberg, Karl, “Covert Channels in Trusted LANS”, *Proceedings of the 11th National Computer Security Conference*, October 1988, Baltimore, Maryland, pp. 226-232.
- 10.10 Fellows, Jon; Hemenway, Judy; Kelem, Nancy; and Romero, Sandra, “The Architecture of a Distributed Trusted Computing Base”, *Proceedings of the*

10th National Computer Security Conference, Gaithersburg, Maryland, September 1987, pp. 68-77.

- 10.11 Ford, Warwick, Computer Communications Security: Principles, Standard Protocols and Techniques, Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- 10.12 Freeman, J. W.; Neely, R. B. and Dinolt, G. W., “An Internet System Security Policy and Formal Model”, *Proceedings of the 11th National Computer Security Conference*, Baltimore, Maryland, October 1988, pp. 10-19.
- 10.13 Gasser, Morrie, Building a Secure Computer System, Van Nostrand Reinhold, New York, 1988.
- 10.14 Herbison, B.J., “Security on an Ethernet”, *Proceedings of the 11th National Security Conference*, Baltimore, Maryland, October 1988, pp. 219-225.
- 10.15 Johnson, Howard and Layne, J. Daniel, “A Mission-Critical Approach to Network Security”, *Proceedings of the 10th National Computer Security Conference*, Gaithersburg, Maryland, September 1987, pp. 15-24.
- 10.16 Kent, S.T., “Network Security: A Top-Down View Shows Problem”, *Data Communications*, June 1978, pp. 57.
- 10.17 Kirkpatrick, Kimberly, “Standards for Network Security”, *Proceedings of the 11th National Computer Security Conference*, October 1988, pp. 201-211.
- 10.18 Losocco, Peter, “A Security Model and Policy for a MLS LAN”, *Proceedings of the 10th National Security Conference*, Gaithersburg, Maryland, September 1987, pp. 25-37.
- 10.19 Millen, Jonathan, “A Network Security Perspective”, *Proceedings of the 9th National Computer Security Conference*, Gaithersburg, Maryland, September 1986, pp. 7-15.
- 10.20 Nelson, Ruth, “SDNS Services and Architecture”, *Proceedings of the 10th National Computer Security Conference*, September 1987, pp. 153-157.

- 10.21 Shaffer, Steven L. and Simon, Alan R., Network Security, AP Professional, Boston, Massachusetts, 1994.
- 10.22 Sidhu, Deepinder and Gasser, Morrie, "A Multilevel Secure Local Area Network", *Proceedings of the 1982 Symposium on Security and Privacy*, pp. 137-143.
- 10.23 Tater, Gary and Kerut, Edmund, "The Secure Data Network System: An Overview", *Proceedings of the 10th National Computer Security Conference*, Gaithersburg, Maryland, September 1987, pp. 150-152.

11

SECURE ELECTRONIC COMMERCE

*If your company operates a high-visibility
electronic-commerce Website, that site probably is
experiencing as many as five serious security
attacks per month*

-- November 20, 1997 InternetWeek

Electronic Commerce refers to the conducting of business between two parties electronically. This entails a contractual relationship between parties who usually have never met each other and have no real idea of who it is they are dealing with. Due to the surge in popularity of the Web, Electronic Commerce is often thought of to be synonymous with transactions conducted over the Internet. Many elements must come together in order for Electronic Commerce to be accomplished but the top two concerns of information technology professionals are the security of the electronic network and meeting customers' and partners' privacy expectations [11.4].

Electronic Commerce does not introduce whole new areas of computer and network security but rather provides an opportunity to apply the many aspects of security discussed in previous chapters of this text. The same elements of security found elsewhere, confidentiality, integrity, availability, and assurance, are also present in Electronic Commerce. In addition to these, however, a new level of trust is required. It is not only important to be assured that an individual is communicating with who they think they are, but also that they can trust that individual, company, or organization they are entering into a transaction with.

The threats to Electronic Commerce are the same as the threats to network security. Applying these to the real-world aspect of the Internet results in seven specific concerns in Electronic Commerce:

- ***Destruction of Data*** – Both data contained on web sites as well as data that is traveling through the network must be protected against accidental or malicious loss.
- ***Modification of Data*** – Data contained on web sites and in transit in the network must be safe from unauthorized or accidental alteration.
- ***Unauthorized Disclosure of Information*** – Data should not be disclosed to individuals not possessing the appropriate permission to access it.
- ***Non-repudiation*** – Both the customer and business must be assured that neither party can deny that an order has been placed and the goods received when a transaction has indeed taken place.
- ***Interference of Operation*** – Data meant for a specific web site should not be rerouted to another location nor should a site be subjected to traffic not intended for it.
- ***Misrepresentation*** – Both parties to an on-line transaction need to be assured that the other individual can be trusted to complete their portion of the transaction in the agreed upon manner.
- ***Inappropriate Use of Data*** – Individuals taking part in an on-line transaction need the assurance that personal data they supply will not be used by others not involved in the transaction or in a manner they had not intended.

Not only are the threats in Electronic Commerce simply a real-world example of the threats to network security, the solutions are also basically the same. In this chapter a number of evolving aspects of Electronic Commerce are discussed along with the methods that security is being addressed for each.

11.1 Certificate Authorities

In a completely trusted environment no mechanism would be needed to ensure that all parties to a transaction abide by the agreement. The Internet, unfortunately, is not a trusted environment. In fact, the Internet is much closer to a highway frequented by all manner of bandits. There is no inherent way to know the identity of any individual you meet, communication can not be considered private as others may be listening at any given time, and transacting any business on the highway is done at considerable risk as you may not receive what you pay for and may not know where to locate the other party again. In order to conduct business on this highway you need the services of a third individual trusted by both parties. This third party can verify the trustworthiness of both sides so that each side feels more secure and confident that they

will receive what they have agreed to. The United States Postal Service (USPS) has been serving as a trusted third party for a number of years. Individuals may go to the Post Office and utilize special mail services, such as registered and certified mail, in order to ensure that there is a certain level of non-repudiation in terms of communication between parties. Other services such as COD (Cash On Delivery) can also be utilized to ensure that goods were indeed received and paid for. A similar trusted third party is required for transactions conducted over the Internet.

Interestingly enough, the U.S. Postal Service proposed in 1996 a series of services to mirror those it offered in its normal operations. These services were to represent the electronic equivalent of the postmark in serving as a time and date stamp for mail sent over the Internet. The postmark also serves another important purpose on the Internet and that is to validate that the document has not been altered during transmission. Other services were also offered which could be used to verify who it was that sent a message or other electronic document. The term generally used to describe this last service is *certificate authority*.

There are two methods that the certificate authority can use to provide this trusted third party function. The one generally associated with certificate authorities is to provide a certificate which has an expiration date associated with it so that it can not be used later by another party to forge the identity of another. Based upon Public Key Encryption, a certificate authority basically stores and supplies upon request the public key for the site an individual wants to communicate with. The certificate authority is trusted by both sides to supply the correct key. The actual process is slightly more complicated because of what is known as a “man in the middle” attack. In this attack another individual, wanting to either intercept or modify the message being sent will intercept the request to the certificate authority and send it’s own public key back at the same time it sends a request for the public key of the intended receiver. The originator then encrypts the message using the imposter’s public key. The imposter can then decrypt this message using the matching private key, read or modify the message, then encrypt it with the receiver’s public key it has received from the certificate authority. To eliminate this potential attack, an additional step is added which entails the certificate authority encrypting the key for the receiver with the certificate authority’s own private key. The originator now can be assured that this came from the certificate authority because it can only be decrypted using the certificate authority’s public key. A number of commercial sites have been created to serve as certificate authorities. Variations on this basic method exist which may include additional information often being sent in the ‘certificate’ to the originator such as an expiration date for the certificate. See [11.5] for a discussion of the X.509 protocol certificate and its components.

Another method to accomplish trusted communication also adds another element to the transaction, a certain level of anonymity. If an individual uses cash to purchase

an item in a store, the proprietor does not need to identify the individual and the transaction can take place with the buyer being anonymous. Anonymizers have been created on the Internet which can take an email message, send it through a couple different servers, then deliver it to the recipient. The identity of the originator can thus be kept secret. The only problem with this is that it is hard to respond to an individual who is anonymous. A variation on the traditional certificate authority is the use of “pseudonomous” servers which provide users with a unique pseudonym that can be used to send messages back to. The servers are trusted by all parties (like regular certificate authorities) to keep an individual’s identity secret and also to make sure that the pseudonym is unique and is only used by a single individual. Obviously anonymous business transactions are hampered by the fact that some means of payment still needs to take place and the item delivered. A method known as Digital Cash can aid in the payment portion of this problem.

An open standard called the Secure Electronic Transaction (SET) utilizes certificate authorities along with public-key cryptography to provide reliable electronic transactions. SET is designed to address the most important elements in electronic transactions:

- providing confidential and reliable data transmission
- authentication of all parties to a transaction
- ensure the integrity of all financial information

It is important to note that SET, which was originally proposed by MasterCard and Visa, is designed to secure data in transit, not to provide an end-to-end security solution.

11.2 Smart Cards

In 1995 there were one billion smart cards in use worldwide with 90 percent of them being used in Europe. By 2001 that figure is expected by some analysts to triple [11.3]. The idea of carrying a card that can be used to identify individuals is not new – we’ve been using credit cards, for example, for years. The problem most often addressed by smart cards is authenticating the identity of the user [11.1]. Cards can also, however, be used as a form of anonymous digital cash as credit information can be stored on them. An example of this digital cash aspect are the common telephone cards that can be purchased from many different stores which come with a variety of credit amounts. Cards used in subways are another example of this type of simple digital cash technology. Newer smart cards, however, are much more versatile than this and

can be used as a much more reliable means to authenticate the identity of users as well as actually maintaining records on the card itself.

In order for smart cards to be able to identify a user they must utilize some form of authentication as discussed earlier in this text. PIN numbers are a common technique used in conjunction with smart cards but while easy to implement it is also the least secure method. Newer cards can utilize various forms of biometrics to identify the user but this additional assurance comes with a higher pricetag. True smart cards (as differentiated from more common cards such as credit cards which contain only a simple magnetic strip) consist of an integrated circuit chip containing a central processing unit, random access memory, and some non-volatile data storage. There is actually a chip operating system used to control access to the information on the card.

Smart cards can be used in many different applications. The Smart Card Industry Association has identified eight areas in which smart cards are being used today. These areas are:

- Loyalty (tracking a user's buying habits)
- Banking
- Healthcare
- Telephony
- Information Technology
- Mass Transit
- Government
- Identification

Visa has produced multiapplication cards which combine several areas such as credit, debit, and loyalty functions. Obviously, encryption plays a large part in the security of any of these smart card functions. Cards that do not contain their own microprocessor rely heavily on the security of the card reader to maintain the security of the information that is stored on the card itself.

11.3 Interesting Applications of Electronic Commerce

There are many possible applications or industries that can take advantage of electronic commerce. The security concerns for most of these will be similar although several have unique problems or issues that surround their application in an electronic environment. (Not mentioned in any of the examples below, but still of concern, is the issue of ascertaining the age of potential consumers.)

11.3.1 Home Banking

There are two basic ways to conduct banking transactions from home. The first is Internet Banking in which individuals can access their bank accounts via the Internet. The second is a direct connection to the bank via a dial-in modem. Either method introduces a level of banking that was unknown before. Accounts can now be accessed 24 hours a day, 7 days a week as opposed to individuals being constrained by the bank's hours of operation. The range of operations available are impressive including the ability to:

- Balance checkbooks
- Determine the status of checks
- Verify deposits and determine current balance
- Pay bills automatically
- Transfer funds between accounts
- Reorder checks
- Stop payment on a paper check
- Apply for a loan

Internet banking also provides new opportunities to “shop around” for a bank that were not available before. In the past individuals were basically limited to the local bank for banking services. Today banks worldwide are available thus increasing competition and creating a better environment for the consumer. Security concerns include confidentiality, integrity, reliability, and assurance. Encryption is heavily relied upon in the home banking application of electronic commerce.

11.3.2 Stock Brokerages

The Internet has not only changed the way a lot of folks conduct business, it has also changed the way that many invest in the stock market. The availability of information on the Internet along with the speed that it can be obtained has enabled countless individuals to become part of a community once reserved for a select few in the business world. Anybody can now become a “stock trader” and dabble freely in stock speculating. The nature of on-line stock trading has, however, raised a few issues. First there is the obvious security issues. There is no real difference here than in other on-line transactions as individuals are concerned with transaction integrity, confidentiality, and assurance. Verification of transactions and non-repudiation are a big issue as delays or loss of an order to sell or buy can mean a tremendous loss or gain. In a speech to the National Press Club, Arthur Levitt, Chairman of the U.S. Securities & Exchange Commission specifically addressed issues that were of concern to the “day

traders” – individuals whose time horizon for moving in and out of stock positions is measured in minutes or even seconds [11.2]. Levitt explained that, while the Internet makes it seem that an individual has a direct connection to the stock market, the connection is still to a broker who then sends it to the market. Delays and breaks in the system may occur and electronic lines form to both sell and buy stocks. If an individual’s order is buried in a long line of orders being processed for a particularly active stock, the price may radically change by the time the order finally reaches the front of the line. Levitt encouraged individuals to utilize limit orders – orders that place a level on the price which a buy or sell order will not exceed. Levitt also cautioned firms providing on-line services to keep pace with their customer base. He stated that 15,000 new accounts are being opened daily, an astounding number of new customers for systems to handle.

11.3.3 Gambling

The most controversial new on-line offering is the introduction of on-line gaming (gambling). On-line gambling has the same security concerns as other on-line electronic business transactions. Typically the way that on-line gambling is conducted is to either download software offered by an on-line casino or to play the games offered on the web (the downloaded versions generally offer a better selection of games and higher quality graphics and sound). After deciding which game to play an individual has to purchase chips/credit from the casino. This is accomplished by opening an account generally using a credit card – although, the gaming industry being what it is, cashier’s checks, wire transfers, bank drafts, and personal checks are also accepted. When an individual is done playing they are given the option of leaving whatever balance they have in their account or “cashing out” and having a check sent to their home address. Security is provided using encryption for credit information and a userid/password combination for access to an individual’s account. One interesting issue in on-line gambling are several lawsuits filed for individuals who claim that they cannot be required to pay their credit card bills because gambling debts are uncollectible. The original suit was brought by an individual from California who had run up \$70,000 in debt on her credit cards.

11.3.4 The Shopping Mall

Imagine having the ability to shop from literally thousands of stores instead of the few dozen found at most suburban malls. This is what on-line shopping malls bring to electronic commerce. Instead of being limited to the stores found at local shopping centers, on-line malls make stores from around the world available to shoppers. Payment methods vary and the security required is similar to the other

elements of electronic commerce. The concept of a shopping cart has even been introduced to these electronic malls so that a consumer can visit several locations, selecting various items for purchase and then making a single transaction instead of individual transactions for each item purchased.

11.3.5 Other Applications

There are many other applications of electronic commerce. Two final applications that have been somewhat revolutionized by the Internet are electronic publishing and on-line travel agents. On-line travel agencies allow consumers to purchase tickets online (in fact the electronic ticket has now become common in the transportation industry), reserve rental cars, make hotel reservations, and book cruises to a variety of exotic locations. One can easily imagine the privacy concerns involved in this application of on-line commerce. On-line publishing has opened up a whole new avenue for those who wish to have their viewpoints displayed or for special interest groups who don't have a large membership and would normally not have an outlet for their material. Most major publications now offer on-line versions. Sometimes these are streamlined versions of their printed publication and are available to everyone. In other cases there may be special articles that are only seen in the on-line version. Some on-line publications are only available through subscription. A concern to all, especially to subscription services, is the copying of materials on-line. Many sites may wish to provide links to certain articles found on another organization's servers or make a copy to place on their own. Copyright laws apply equally to material in this medium as they do in the print arena.

11.4 Digital Cash

Digital Cash is a form of electronic money which also includes electronic checks and credit cards. There are two broad methods to handle digital cash transactions, off-line and on-line. Off-line digital cash allows the users to conduct business without having any direct connection to a bank. On-line digital cash interacts with a bank to determine if sufficient funds are available. In discussing either method there are four major concerns that must be addressed:

- Security
- Anonymity
- Reliability
- Scalability

These security concerns and the approaches to addressing them are fairly obvious. Encryption is used to avoid eavesdropping on or modification to electronic transactions. Digital signatures and time-stamped certificates are used to validate the identities of all parties and thus avoid the possible spoofing of transactions. In addition to these threats, however, digital cash must also be protected against the digital counterpart of counterfeiting. Counterfeiting paper money is an involved and complicated process, creating electronic patterns contained on a card is not. Digital cash systems must be able to differentiate between authentic and forged digital cash. Another security problem with digital cash is the issue of double spending – the ability to copy and duplicate the digital cash. For off-line systems, sophisticated smart cards containing a tamper-proof chip can be used. These systems maintain a database of all transactions and amounts on the card itself. Attempts to modify the database will normally result in the destruction of the information on the card thus making it unusable. An easier approach is to utilize on-line systems which would simply keep track of digital cash that has already exchanged hands. While the on-line approach simplifies the double-spending detection process, it suffers from the disadvantage of a loss of anonymity.

The biggest advantage to cash transactions is the fact that they can be done anonymously and are untraceable. This is a tremendous advantage for those who are concerned with their privacy. On-line systems can keep track of spending patterns for individuals and records will exist detailing what purchases have been made. Hopefully this information will be kept confidential but there are times information may become public an individual may have wanted to be kept secret (the video rental habits of an individual running for an elected office, for example). Without the guarantee of anonymity, digital cash systems lose an important characteristic that would distinguish them from other forms of electronic transactions. Off-line systems hold the greatest promise in providing this desired anonymity.

The last two concerns about digital cash are reliability and scalability, both of which must be present for the concept of digital cash to catch on. Consumers need the assurance that their money in digital cash form will be there when they want it (i.e., will not have accidentally been erased or modified) and that they can use the system when they want to (i.e., that retailers will have compatible systems that will recognize their digital cash system. The international nature of the Internet presents some interesting concerns with digital cash since the value of a nation's currency fluctuates constantly when compared with another nation's currency.

11.5 Trusting the Web

Much has been written about both the potential of web-based sites for electronic commerce and the problems that web sites continually experience in terms of security.

While many people are currently accessing and using web sites to participate in electronic commerce, many more have held back because of either real or perceived risks associated with conducting electronic business. The major concerns are basically security and privacy. People are worried that information that they disclose will become public or be used in a manner they did not wish. To address these concerns two organizations have been created which provide a trusted third party role similar to that provided by certificate authorities. These organizations provide a “stamp of approval” which the site can display to indicate that they are abiding by certain strict guidelines the approving organization considers to be critical for trusted web-based business transactions. These guidelines generally cover three areas of concern:

- ***Ethical Business Practices*** – consumers want to be assured that the company they are dealing with is reputable, will be able to meet its contractual obligations, and abides by standard business practices in terms of warranties and return of goods.
- ***Transaction Integrity*** – information provided by the consumer should be safe from interception, modification, or deletion by an unauthorized individual or through errors in the system or transmission itself.
- ***Protection of Privacy*** – a considerable amount of information is gathered by web sites and consumers want to be assured that information they provide will be kept confidential and will be safe from unauthorized or accidental disclosure.

The approach is fairly simple. Consumers that wish to conduct business on the web should in theory look for one of the stamps or seals of approval granted by these organizations on the site they are considering doing business with. If the seal is not found then they are accepting the inherent risk that accompanies any transaction between unknown parties. If the stamp or seal is found then they are assured (there are ways to verify the stamp or seal is authentic) that the site has met certain guidelines and is believed to be trustworthy by the granting organization. The organizations that have been created to address these issues and that provide the stamp or seal of approval conduct both initial and periodic reviews of sites wishing to obtain their stamp or seal. The industry is also encouraged by the organizations to police itself and report any sites which bear one of these marks that has violated privacy policies or that has misused the stamp or seal.

11.6 Summary

Electronic commerce has not introduced a tremendous number of new security concerns but rather has created many real-world applications which require efficient and effective security. The security concerns in electronic commerce are basically the same for the networked environment discussed in the previous chapter. This should not come as a surprise since electronic commerce generally refers to the conducting of business over the Internet which is nothing more than an extremely large collection of networks. There are several very interesting topics in electronic commerce, however, including determining methods to securely create and use anonymous digital cash and methods to provide trust in web sites. Paramount in every application of electronic commerce is the issue of privacy and trust. Without these electronic commerce would not be able to survive. With them electronic commerce will continue to expand.

11.7 Projects

- 11.1 The Internet is an ever-changing environment. What is the current offering of the United States Postal Service in terms of certified e-mail and as a certification authority?. What other commercial companies offer public certification authority services? Which of the seven threats discussed earlier in this chapter do these services address?
- 11.2 Similar to problem 11.1, what commercial companies offer digital cash services? Which of the seven threats do these services address?
- 11.3 What techniques are currently being used to prevent ‘counterfeiting’ in digital cash systems? What current commercial systems are available that offer digital cash services? Are these on- or off-line systems? How do they handle the issue of anonymity?
- 11.4 Describe how to implement an international (i.e., must be able to handle different national currencies) off-line digital cash system. How would you address issues of large cash transactions of the type that currently must be reported by banks?
- 11.5 What organizations can you find that address the issue of web trust? What are the differences between the certification process for sites wishing to obtain certification from one of these organizations?

- 11.6 Only a few interesting applications of electronic commerce were discussed in this chapter. What other applications are there? Do any of them have peculiar or unique security concerns not mentioned in this chapter?
- 11.7 What is the current status (or what were the results) of the on-line gambling debt lawsuits (check class action filings in Alabama and Washington as well as the case of Cynthia Haines in California)?

11.8 References

- 11.1 Denning, Dorothy E and Peter J., Internet Besieged: Countering Cyberspace Scofflaws, Addison-Wesley, Reading, Massachusetts, 1998.
- 11.2 Levitt, Arthur, “Plain Talk About On-line Investing”, speech given at the National Press Club, May 4, 1999, transcript available from <http://www.sec.gov/news/speeches/spch274.htm>, July 17, 1999.
- 11.3 Pine, Jordan T., “Smart Cards: Bringing Smart Business to the Masses”, available from http://www.hemnetcom/E-Business/Feature/body_smartcards.html, July 17, 1999.
- 11.4 Robinson, Teri, “Reinventing the Business Wheel”, *InformationWeek*, June 21, 1999, pp. 6SS-10SS.
- 11.5 Schneier, Bruce, Applied Cryptography, John Wiley & Sons, Inc., New York, NY, 1994.

11.9 Extended Bibliography

- 11.6 Berinato, Scott and Michael Moeller, “Big boost for smart cards”, *PCWeek*, July 14, 1997, available at <http://www.zdnet.com/pcweek/news/0714/14card.html>, July 19, 1999.
- 11.7 Denning, Dorothy E., Information Warfare and Security, Addison Wesley, Reading, Massachusetts, 1999.

- 11.8 Gillmor, Steve, Jeff Angus, and Sean Gallagher, "New Model for E-Commerce", *InformationWeek*, June 28, 1999, pp. 65-74.
- 11.9 Grabbe, J. Orlin, "Concepts in Digital Cash", available from <http://www.aci.net/kalliste/digiprin.htm>, July 2, 1999.
- 11.10 Guida, Richard, "PKI holds the key to making EC secure", an interview with Richard Guida, *Government Computer News*, June 14, 1999, page 16.
- 11.11 Internet Gaming News, "Credit Card Companies Are Sued", *Gaming Biz OnLine*, June 22, 1999, available from <http://www.gamingbiz.com/net22.html>, July 2, 1999.
- 11.12 Levitt, Jason, "In Keys We Trust", *InformationWeek*, June 14, 1999, pp. 75-86.
- 11.13 Neshevich, Carol, "One e-commerce standard called unrealistic", *Network World Today*, June 11, 1998, available from <http://www2.idg.com.au/nwwdb.NSF>.
- 11.14 Rothke, Ben, "Is SET NeXT?", *Information Security*, July 1999, page 12.

12

WORLD WIDE WEB (WWW) SECURITY

Over the last ten years the Internet has grown from being a small research-based conglomeration of computers and networks to a major communication medium. Where it was once very rare for a company to have a web page, companies now have multiple pages. Not only do major corporations have pages, but most every organization, club, group, and individual has a page as well. Furthermore, a typical personal resume now has a home page address listed below the home phone number and physical home address. Unfortunately, with this increased activity comes the usual growing pains; the availability of private or confidential information, computer espionage, malicious users wanting to break into the various web sites for one reason or another, and the unscrupulous inside-trader looking for potential business advantages associated with closely held information. With all of the growth and all of the newly available information that is now on out web pages, we must ask ourselves “*how can I protect my web site?*”

The amount of security a web site requires depends on the purpose and functionality of a site. A simple personal web page that lists the owner’s hobbies and interests will have substantially different requirements than a site devoted to electronic commerce or banking and exchanges private information with the user. Let’s not also forget that today’s complex web pages are capable of downloading and running programs on the user’s computer. What type of security do you suppose the casual person should employ before browsing the web? Something needs to be done to prevent these complex web pages from installing or running malicious programs.

This chapter addresses many of the security issues associated with the World Wide Web (WWW), or simply ‘*the web*.’ It begins by looking at some of the basic security issues associated with the most common browser software. The discussion will continue with an examination of the programming techniques used to create the complex web pages and how they impact the user and programmer. More advanced

topics, such as web server configuration and secure communication will be presented, followed by a discussion of the interaction between a firewall and a web server.

Because many of the terms associated with web activity can have multiple meanings some terminology needs to be agreed upon at the onset. To begin with, *browser* will refer to the software packages (i.e., Internet Explorer and Netscape) as opposed to the person that is rummaging through the web. This person, however, will be referred to as *the user*. Finally, the organization or person responsible for the content and maintenance of the web server will be called *the owner*.

12.1 Browser Security

While the ratios may be changing, the number of people exploring, or surfing, the web is significantly greater than the people creating web pages to surf. Even though these users may be malicious, they too can be the victim of an unscrupulous web site owner. If the user's browser is not appropriately configured, it can allow the owner to read files, write files, erase files, upload information, and execute commands. The weary user therefore needs to know about the risks of surfing and how to protect themselves (safe-surfing?!).

12.1.1 User Certification

A user certificate, or personal certificate, is a unique digital identification that can be presented to other users or web sites on the Internet. The concept of a personal certification relies heavily on public key cryptography (see chapter 12) and employs both a public and private key. As could be expected, the certificate can be used to encrypt and decrypt e-mail, digitally sign e-mail, and uniquely identify one's self to a web site. With the numerous benefits and possibilities associated with user certification, one would expect to see it widely used. This is not the case, however, and user certification is only used in limited quantities; most commonly as a means of access control in corporate Intranets.

The issues preventing user certification from widespread use are twofold. The first concerns the security of the system and software protecting the unique portion of the key maintained by the user. In public key encryption, the user maintains a private encryption key and a public server contains the complementary public key. In the event that a person's private key was to become compromised, another person could potentially masquerade as that user. While the unauthorized user would still need a password to activate the key pair, the existing state of security on the Internet makes this a not so implausible idea. The second concerns the security of the software

manipulating the key halves, specifically the browsers. It seems that every week a new security hole is discovered in the browsers. Until browser technology and programming improves, the adoption of user certification will be slowed down.

Despite the weaknesses in the user certification process and infrastructure, many people think that personal certificates will be used in the near future as legally binding electronic signatures [12.2]. These people would be quite correct because the fundamental concept can be found in the well-known security concept of **PKI, public key infrastructure**. More information can be found on PKI in chapter 14.

12.1.2 Cookies

Developed by the Netscape Corporation, a “cookie” is a mechanism to provide a sort of memory to the browser process. Normally, every browser request is treated as an independent action distinct in every manner from the previous or next action. The fact that a page request may be one in a series of requests is lost on the browser. This makes it difficult to remember information from page to page, such as shopping carts, travel itineraries, and even user names and passwords. Cookies solve this problem by acting as a notepad and recording information that can be accessed by multiple pages.

A cookie is a small piece of information (at most 4K bytes), typically containing information that is site-specific, that is accessed with every page request from a given web site. For example, a cookie from <http://www.genericwebsite.com> would be accessed with every page request to the genericwebsite web site. This usage allows the server and browser to thread multiple page requests into what appears to be a single user session. The contents of a cookie include information indicating the associated web site with which to interact, which specific pages on the web site to interact, and the particular information to share with the site. This leads to many interesting questions regarding security and unauthorized access to the information in a cookie associated with another web site.

12.1.2.1 Cookie Privacy

The existing process used to examine and manipulate cookies prevents them from being accessed by web sites other than their originating site. Thus, cookies cannot be the means in which information about you or your computer system is stolen. Additionally, they can only store information that the user has provided to a web site. For example, if you provide an on-line store site with your address and phone number, the web site can enter this information into a cookie on your computer. The next time you access the web site, your browser will send the cookie to the web site and it will be automatically included in your next order. While this is very convenient, it presents an interesting privacy issue, and an advertiser’s dream.

Each time a user accesses a web site it can potentially leave behind a significant amount of information about the user. Examples of this include the user's IP address, the brand and version of the user's browser, the brand and version of the user's operating system, and the web site the user last accessed. This information, when collected from multiple web sites, can be used to create a personal profile of interests and habits. Cookies make this process possible without having to combine the information contained in numerous web site logs. Furthermore, this can be done without violating the security rules regarding unauthorized access to cookies. One particular advertising service does just this with the assistance of the web site owners. Each owner subscribes to the service and provides both an ad and a link to the service from its own pages. The link is used to display an advertisement on the owner's page. The browser not only follows the link to obtain and display the advertisement, but to access a cookie on the user's system that identifies the specific user to the service. Note that both of these actions are invisible to the user and could just as easily not have happened. The service then knows the source of the call (the owner's page) and if the user chooses to click on the advertisement and go to the advertiser that site information as well. The end result is that the service can now track the interests and hobbies of a single user. For example, when Doreen accesses www.sodapop.com the site loads an image it obtains from www.adservice.com. The ad service site not only displays the advertisement for www.snackfood.com but reads the unique user number it has place on the user's system in a cookie that only the ad service can access. The ad service now knows that the user is interested in soft drinks. In the event that the user clicks on the advertisement for snack food, the ad service will know that user is interested in snack food as well. Eventually, the ad service will know quite a bit about Doreen's interests. In the best of cases, this information will only be used to advertise products that Doreen will find interesting. In a less benign situation, the information can be used to identify the user with a reasonable degree of accuracy, especially since the IP address and possibly the user's e-mail address are obtainable. Many people find this an invasion of privacy and very unnerving. Regardless of how a user views it, cookies are a significant threat to user privacy. Keep in mind, however, they are not the only threat.

12.1.2.2 Cookies and System Security

Many web sites use cookies not only to contain information about a session, but for security purposes as well. For example, many sites that request a username and password can keep that information in a cookie, thus eliminating the need for the user to enter the information with each visit. This, of course, can be a significant vulnerability if it is not implemented in a secure fashion. If this information were passed or stored in clear text, compromise of the network (with a sniffer) or the system,

could easily result in compromise of the access controls. The end result of this be unauthorized activity such as bank transactions or on-line purchases.

Web designers that use cookies for user authentication should be aware of the potential security vulnerabilities and plan accordingly. Cookies should always contain as little private or personal information as possible. Additionally, they should never contain sensitive information, such as user names or passwords, in an unencrypted format. Designers should also use mechanisms to verify that the person using a specific cookie is authorized to do so. A popular cookie design scheme is to include the following information in a cookie [12.4]:

- Session ID
- Authorization information
- Time and date of creation
- Expiration time and date – to reduce the number of potential compromises or exploits. Once a cookie expires, it cannot be used by anyone, regardless of authorization.
- IP address of the authorized user – to limit the usage of the cookie to an authorized system. While an IP address can be spoofed, it is both rare and difficult.
- A message authenticity check (MAC) code – a checksum of sorts to verify the integrity of the cookie.

For the more sensitive applications, a secure protocol such as SSL should be used to encrypt all communication between the user and the web server. This will reduce the likelihood of network eavesdropping with sniffers. More details about this can be found below in the protocol discussion.

With all of the possible privacy and security issues that exist, a user may not wish to receive cookies from a web site. Most, if not all, browsers that exist today allow the user to either be notified when they are about to receive a cookie and decline it, or simply not accept any cookies. The drawbacks of this are repeated requests to accept cookies from the web site. Some browsers allow the user to accept cookies that remain only for the duration of the session and reject persistent cookies that remain for an extended period of time.

12.1.3 User Privacy

There are many issues resolving privacy beyond cookies. These issues include document request privacy, local system and network privacy, and server-side logging. Each of these issues can be used to gain information about the user and lead to unauthorized and unwanted activity. This information can not only be used against the

individual but an organization as well. With user information, passwords, and site access history, a malicious person (or group) can gain enough information about an organization to launch a successful attack. For example, knowing an e-mail address, IP address, password for some other web site, and a history of previously viewed documents, a person could understand what the user network looks like (from data regarding previously viewed web sites), and a potentially valid password for the user's network (most people use the same password everywhere).

Most web sites, when accessed by a user, log the IP address and computer host name. Some web sites also identify the web page you were previously viewing. In an ideal world, this information would be used for statistics and improving the web site or for debugging purposes. In that same ideal world, only authorized administrators would have access to the log files. In many cases, however, this information is used for less than desirable purposes (e.g., mailing lists or sales calls) or is readable by anyone with system access. In some cases, the information gathered may be significantly more than e-mail addresses. When a user fills out a form, it may include this information in the web address. The next web site visited by the user may then gather this information for its own purposes. For example, a user fills out a request for more information that includes their street address and phone number. The user then decides to look at one of the many hacker web sites (or maybe an "adult" web site). If that web site is logging user history information, they will have this person's contact information. In the event it is a hacker web site, the page may do more than just obtain contact information.

Some web sites can instruct a web browser to providing network login information. This includes user names and encrypted versions of the associated passwords. Once this information is known, access to the user's organization can be very easy. This may be little more than a concern for home-users, but a significant one to anyone or any company that has proprietary information. Once this information is obtained, however, the password can be brute-force attacked and the account compromised. While this type of attack does not work on all browsers, or at least many of the latest web browsers, it is still of significant concern. Users may think this is not a significant problem because there is no access to the network through the firewall. Depending on the configuration of the firewall and the company network as a whole, this may or may not be the case. With the complexity of today's firewalls and networks, these security issues cannot be overlooked. This now begs the question, *"how do I prevent this?"* The answer is twofold. First, use a strong password. Every organization defines this differently, but consider a password that is at least 8 characters long and uses both numbers and letters intermixed. Second, obtain the latest version of your web browser. Most current web browsers can be configured to prevent this type of vulnerability. While the user is at it, he or she should make sure they are using network file sharing in a secure manner.

12.2 Scripts

Scripts are programs designed to perform detailed tasks on the user's computer. When a user accesses the web site, the page is automatically downloaded to the user's computer and the browser executes the program. There are two common types of script programming languages, ActiveX and Java. Another type of web-based program is called a **Common Gateway Interface** (CGI). CGI files can be written in one of many programming languages. The key difference between the other scripts and CGI is that the user typically activates CGI scripts by clicking on a button or picture, and they execute on the web server. This can lead to many server-side vulnerabilities, especially when they are poorly designed and written. Each of these is discussed in more detail below.

12.2.1 ActiveX

ActiveX is a scripting language that makes it easier for web developers to create unique, interactive applications and web sites. There are more than 1,000 ActiveX controls readily available in the public domain, allowing developers to reuse many of the interactive parts of their Web sites. ActiveX can be used with a wide variety of programming languages, including Java, Borland Delphi, Microsoft Visual Basic, Microsoft Visual C++, Borland C++, and others.

Microsoft developed ActiveX for distributing software over the Internet. ActiveX objects, called **controls**, are the interactive graphical items found on a web page (e.g., a marquee, a list box, and sounds). Unlike some of the other scripting languages, ActiveX controls are executables and therefore distributed in precompiled format for each operating system. The files are used throughout Microsoft Windows and have the file extension “.OCX”.

One of the security concerns of ActiveX is that there are no restrictions on a control's action. Security is partially controlled by the author and a third party. The author submits the finished control to the third party for a digital signature. In doing so, the author also signs a statement indicating that the control is free from viruses or any other malicious code. The third party digitally signs the control such that it cannot be modified (with a checksum) without notice. Before the control is downloaded to the user system, the user must first authorize the download and the digital signature is authenticated. While this process does slow down the spread of dangerous controls, it does not guarantee total protection. In fact, if the user's system does go down, little more than sending a nasty email to the developer can be done. If the user attempts to load a control that has not been signed by a known and trusted authority, the browser will warn the user and prompt for acceptance or cancellation.

The greatest threat associated with an ActiveX control is less-obvious activity that could occur without the knowledge of the user. For example, an ActiveX control could send personal or confidential information from the user's computer over the Internet. This type of subtle activity could be difficult to detect and thus difficult to correct. Furthermore, the limited activity logging of some browsers will make identifying the dangerous control very tricky. Some security authorities recommend avoiding this and the other ActiveX issues by either denying all ActiveX control downloads or at least requiring the browser to prompt the user before download.

12.2.2 Java and JavaScript

Sun Microsystems answer to ActiveX is **Java**. Java is a freely available object oriented programming language. Java started as a programming language for embedded controllers, such as those found in ovens and other kitchen appliances. According to Sun, "the initial design goal for Java was to produce a language that was simple to use, be familiar on some level to other languages programmers were familiar with (in this case, C and C++), and portable across hardware platforms" [12.8]. It has since become a widely-used platform-independent programming language for application and software developers in general. Java programs, or *applets*, are most commonly found in HTML documents. They are stored on the web server and downloaded for execution on the user's computer when the associated web pages are loaded into a browser. While Java programs are commonly called *scripts*, they are not to be mistaken for JavaScript.

JavaScript is a set of extensions that increase the functionality of the more common browsers. JavaScript is often used to open and close windows, change browser settings, and even download and start Java programs. Unlike Java, JavaScript is an interpreted language and not distributed as compiled programs. This does not lessen the potential threats and vulnerabilities in JavaScript. As a matter of fact, Java and JavaScript are both known to have security holes.

12.2.2.1 Java Security Weaknesses

Java provides several protection mechanisms to limit the threat posed by the execution of unknown programs on the user's computer. One of these is the limitation of back-end system access such as system command execution, device driver access, and system library loading. Java scripts are also limited to access files in a specific and predetermined directory. This prevents them from accessing and modifying critical system files or confidential user files. Control mechanisms are also in place to limit script network activity. Java scripts are only allowed to communicate with the server that uploaded them. This helps limit the unauthorized distribution and broadcast of

information to third party, and possibly unknown, servers. Network communication, however, can only occur at the expense of local disk access. Java only allows scripts to communicate with the network, if they do not communicate with the user's local disk (and vice-versa). This provides additional protection from data theft and modification. Unfortunately, these controls are not enough to protect the user.

Since its release, a number of security holes resulting from programming bugs have been found and subsequently fixed. Many of them are listed and described here. This list is only provided as a sample and is not intended to be complete—Vulnerabilities that are not publicly known may currently exist and others may be discovered in the future.

- ***Execution of Machine Instructions*** - A bug was found in Java that allowed the creation of an applet that could execute arbitrary machine instructions a file on a user's local disk. This bug is executed by first downloading a binary library file to the user's local disk using the Netscape Navigator caching mechanism. The Java interpreter is then loads the library file into memory and executes it. This bug was fixed starting with Netscape Navigator version 2.02.
- ***Network Connections with Other User Systems*** - Early versions of Netscape Navigator (pre-2.01), allowed communication with other servers on the user's local area network. This could allow the server to potentially obtain data from other systems on the user's network and transmit it back. This data could include both sensitive information and network topology information. This exploit took advantage of trust relationships within the LAN and false Domain Name Service (DNS) information.
- ***Theoretical Security Bypass*** - This theoretical bug would allow the applet to bypass the Java Security Manager and execute forbidden operations. This vulnerability is considered theoretical because no actual exploitations have been reported. The bug was present in the Java Developer's Toolkit v1.1, Microsoft Internet Explorer versions 3.01 and earlier, and in Netscape Navigator 3.01 and earlier. It has since been corrected in more recent versions of each of these packages.

12.2.2.2 JavaScript Security Weaknesses

Like Java, JavaScript also has its checkered past of security vulnerabilities. These vulnerabilities, however, are more privacy related. Listed below is an assortment of JavaScript vulnerabilities, all of which have been correct in the most recent versions of the various browsers and languages.

- ***File Theft*** - Microsoft Internet Explorer 4.0, 4.01, and beta-5.0 was vulnerable to file theft via JavaScript. Explorer allowed JavaScript programs to cut and paste text into file upload fields, thus allowing a web page to obtain any file on the user's disk.
- ***The Netscape Cache Bug*** - Netscape Navigator versions 3.04, 4.07, and 4.5 allow remote sites to read a list of previously viewed web site addresses from the browser cache. This bug only affects Microsoft Windows systems and is described in detail in a Netscape Security Note.
- ***Obtain a User's E-mail Address and Other Preferences*** - Netscape Navigator versions 4.0-4.04 allows a web page that uses JavaScript to open the preferences file and obtain all information contained within it. This can reveal the user's e-mail addresses and details about the user's network. There is potential for this vulnerability to reveal the user's e-mail password. In the event this is the same password for the users network access, network access controls will be compromised.
- ***User File Access*** - Microsoft Internet Explorer 4.0 is vulnerable to web pages that allow the remote web site to access any file on the user's computer. The web developer uses JavaScript to display a 1-pixel by 1-pixel sized frame. The JavaScript associated with the frame scans the user's machine and network disk drive shares for certain files and then sends them to another site on the Internet. This vulnerability does not modify or damage does not allow for file modification, only theft.
- ***Unauthorized File Uploads*** - A vulnerability in the way Netscape Navigator versions 2.0, 3.0, and 3.01 handle JavaScript programs allows the browser to upload files to the user's computer without the user's knowledge or permission. The user may be warned in advance if the "Warn before Submitting a Form Insecurely" option has been selected. The warning will not occur if all communication is through a secure (SSL) channel.

12.2.3 CGI

Another type of scripting tool, or program, is called a *Common Gateway Interface*, or CGI. CGI is a generic term used for a program that resides on the web server as opposed to the user's system. This key difference enables the script to perform some advanced operations such as form handling and advertisement switching. The server also has the ability to redirect data to any e-mail address, to maintain data, to adjust the viewing content to suit the user, and to do other dynamic page creation activity that a browser cannot do on its own. Development of these advanced scripts can be in any language, however, a majority of the CGI scripts written today are in Perl and C. The specific CGI implementation details are left for the reader to research.

Like the other types of scripts, CGI scripts are executable programs that present an opportunity for exploitation. The most significant difference, however, is that a majority of the weaknesses impact the web server as opposed to the user. Because the programs execute on the server, and not on the user's computer, exploits will compromise and impact the server. The operations and the design of CGI scripts often reveal excessive information about the server, the operating system, the layout, or the network. Additionally, poor programming has been known to lead to system compromise. Some of the most famous web server attacks result from developers writing faulty programs that allow the user to execute commands on the remote web server.

One means of protecting a web server from attack is with the use of *CGI wrappers*. CGI Wrappers are scripts that are specifically designed to perform security checks on the primary CGI script. These checks include verifying the ownership of the CGI process and restricting the activity of the script. Two common wrapper programs are *sbox* and *cgiwrap*. The Apache web server also comes with a CGI wrapper called *suEXEC*. Regardless of the source or use of the wrapper, nothing prevents exploitation better than good programming and testing.

12.3 Protocols

To understand the operations and potential security vulnerabilities of the Internet, an understanding of the more common protocols is necessary. The three primary protocols in use today are *HTTP*, *SHTTP* (or *S-HTTP*), and *SSL*. The most common of the three, HTTP, is used for normal web server/browser communication. HTTPS and SSL are used when secure communications are required, such as for electronic commerce.

12.3.1 HTTP

The Hypertext Transfer Protocol (HTTP) is a basic application level protocol used for communication between web servers and browsers. While the primary purpose of HTTP is for communication on the world wide web, the protocol is generic enough that it may be used by many programs for more than just web-based documents. Two common uses for HTTP are program documentation and database front engines.

The HTTP protocol is very simple. The user (or client) sends a request to the server and awaits a response. Upon receipt of the request, the server generates a response (the source code for a web page) and transmits it back to the user. The server then closes the connection. This process is called *stateless*. The server treats all requests as unique and unrelated communications. There is no HTTP mechanism to identify what the client and server last communicated, so there is no condition or “state” associated with the user’s activity. The stateless nature of HTTP allows the creation and use of *proxies*. A proxy is a server that acts as an intermediary between the user and the server (and possibly other proxies). A proxy simply forwards transmissions on to the designated recipient. If the proxy has a cache capability and the requested page is in the cache, it may answer the request without contacting the server.

A part of the client request to a server is a *method* format. The method identifies the general operation that the client is requesting from the server. There are three types of methods: (1) GET, (2) HEAD, and (3) POST [12.1]. The GET method requests the server return the information as contained in a specified data file or described for back end analysis, such as for a database query. The HEAD method is identical to the GET method, except that the server response is only the header information and not the body of the document. This is primarily used for testing. The final method, POST, is used to send and receive information as presented in a specified format. The format is often a form, a news posting, or a database communication.

More details and up-to-date information regarding this protocol can be found on the World Wide Web Consortium web site at <http://www.w3.org>.

12.3.2 SHTTP

SHTTP (secure HTTP) was developed by Enterprise Integration Technologies in response to requirements established by CommerceNet, a coalition of businesses interested in developing the Internet for commercial use [12.7]. SHTTP is a secure extension to the HTTP protocol. Its primary purpose is to protect individual transactions. It provides confidentiality, integrity, and authentication with the use of two message formats: PEM (Privacy Enhanced Mail) and PKCS #7 (Public Key Cryptography Standards) [12.7].

The PEM standards, found in proposed Internet standards RFC 1421 and RFC 1422, define a secure messaging protocol and a supporting public-key infrastructure. The security messaging protocol was designed to support only the most basic of message protection [12.3, 12.5]. It provides security by placing the unprotected communication within a PEM communication. This new PEM-wrapped message is then transmitted with the normal communication mechanisms. The RFC documents that describe the PEM standard are exceptionally detailed and left for further research by the reader.

PKCS #7 is one of many standards developed around 1993. Standard #7 defines the data structures and procedures for digitally signing and encrypting data. It is not only used for SHTTP but for the secure version of MIME called *S/MIME* [12.7]. Within both SHTTP and S/MIME, the following three PKCS #7 data structures are employed: (1) signed data, (2) enveloped data, and (3) signed and enveloped data. Signed data types include a digital signature with the encrypted data content. Enveloped data includes a symmetrically encrypted version of the data along with the appropriate public key for each recipient. Signed and enveloped data is a combination of the first two data types.

The details of PEM and PKCS #7 can be found in their associated Internet standards documentation. These details, however, will be of limited value for the casual web developer. Those interested in the nuances of communication protocols or the mathematical details of the protocols should review the standards. For those not so inclined, the basic understanding provided here as well as a review of employing SHTTP in web development will provide the necessary information to use SHTTP.

12.3.3 SSL

SSL (Secure Socket Layer) is the scheme developed by Netscape Communications Corporation. It is a high-level encryption scheme used to protect transactions in any of the TCP protocols, such as HTTP, NNTP and FTP [12.6]. SSL is widely used on the World Wide Web and identifiable by URLs that begin with “*https://*” (not to be confused with SHTTP URLs beginning with “*shttp://*”). SSL includes provisions for server authentication, client authentication, confidentiality, and integrity. SSL can be broken down into two sub-protocols that identify how the data will be protected and how it will be communicated.

The SSL protocol has two components, the SSL Record Protocol and the SSL Handshaking Protocol. The Record Protocol defines the format for all data items in the communication session. It provides for data compression, integrity checking, encryption, and data packet sizing. This protocol supports changing the protection algorithm at any time during the communication. The Handshaking Protocol is used to negotiate which algorithm will be used for authentication and confidentiality protection.

Supported algorithms include Diffie-Hellman, RSA, and KEA. To support both domestic and international encryption, the exportable implementation limits key length to 40 bits as opposed to the U.S. version that supports 128 bit keys.

It should be noted that this protocol, like the others is every evolving. As such, the Internet standards should be referred to for additional Details.

12.4 Summary

The Internet and the World Wide Web are wonderful places to explore, learn, and waste time on. While you can explore and learn everything about anything it can be a very dangerous adventure. Many of the dangers are a result of uneducated users not taking the necessary precautions. Users should be aware of the different security and privacy issues, such as cookies, Java, and ActiveX. While it would be both annoying and inconvenient to configure a web browser to require a user's permission at every opportunity, care and common sense are required. This is not only the case with users, but with web developers as well. Many of the latest security holes are a result of poor developer programming and planning. In some cases, the security issues are a result of buggy server software. It is in the best interest of the developer and web master to keep up to date with vendor patches and the security community for the latest threat to their environment. It is also in the best interest of the web user to keep up to date with browser technology so as to lessen their risk.

12.5 Projects

- 12.1 Research the latest security vulnerabilities in each of the various web browsers and web servers. Is there a pattern to the issues and characteristics of these vulnerabilities?
- 12.2 How do the security mechanisms of Java and ActiveX compare? Create the design for a new scripting language that attempts to incorporate the best of each security scheme.
- 12.3 Which is the greater threat: a weakness in the security controls or a weakness in the privacy controls? Why?

12.6 References

- 12.1 Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee T., *Hypertext Transfer Protocol -- HTTP/1.1*, <ftp://ftp.isi.edu/in-notes/rfc2616.txt>, June, 1999.
- 12.2 Ford, W., and Baum, M. S., Secure Electronic Commerce, Prentice Hall, Upper Saddle River, New Jersey, 1997.
- 12.3 Kent, S., *Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management*, <ftp://ftp.isi.edu/in-notes/rfc1422.txt>, February 1993.
- 12.4 Kristol, D. M., and Montulli, L., *HTTP State Management Mechanism*, <http://search.ietf.org/internet-drafts/draft-ietf-http-state-man-mec-11.txt>, August 1999.
- 12.5 Linn, J., *Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures*, <ftp://ftp.isi.edu/in-notes/rfc1421.txt>, February 1993.
- 12.6 Netscape Communications Corporation, “How SSL Works”, <http://developer.netscape.com/tech/security/ssl/howitworks.html>, 1999.
- 12.7 Roberts, D., Internet Protocols Handbook, Coriolis Group Books, Scottsdale, Arizona, 1996.
- 12.8 Sun Microsystems, “What is the Java Platform?”, <http://www.sun.com/java /platform.jhtml>, 1999.

12.7 Extended Bibliography

- 12.9 Feghhi, J., Williams, P., and Feghhi, J., Digital Certificates: Applied Internet Security, Addison Wesley Longman, Inc., Reading, Massachusetts, October, 1998.
- 12.10 Garfinkel, S., and Spafford, G., Web Security and Commerce, O'Reilly & Associates, Inc., Sebastopol, California, January, 1997.

- 12.11 Ghosh, A. K., Securing Internet Commerce: Weak Links, Practical Solutions, Wiley, John & Sons, Inc., New York, New York, October, 1997.
- 12.12 Harrison, R., ASP, MTS, ADSI: Web Security, Prentice Hall, Upper Saddle River, New Jersey, November, 1998.
- 12.13 Stein, L. D., Web Security - A Step by Step Reference Guide, Addison Wesley Longman, Inc., Reading, Massachusetts, December, 1997.

13

FIREWALLS

System administrators have found it increasingly hard to protect their computer systems as the number of computers connected to networks has grown. The idea of disconnecting a machine from a network, or a network from other networks, is in opposition to the very reason networks are created, yet many a frustrated administrator has at times wished they could do just that. An alternative, a method to protect the network from outside intruders without limiting the access to the outside world, would greatly ease their concerns. This is the purpose of firewalls. They are intended to limit the damage that can be caused to a network by limiting the access rights of outsiders to the network. This chapter discusses the purpose and design of security gateways and firewalls which have seen a recent rise in popularity.

13.1 Simple Damage Limiting Approaches

A firewall in a building is designed to keep a fire from spreading to other parts of the structure or to at least slow its progress until help can arrive. The fire may burn out of control in one portion of the structure, the firewall simply insures that other portions aren't affected at the same rate. This same idea can be applied to networks and computer systems. If an organization places all of its computing assets on a single network, when a security problem arises the entire network can be affected. This is especially true where access methods such as the UNIX *.rhosts* file (which allows users access to other systems listed in the file without having to provide a password) are available. Often, local machines are configured to implicitly trust each other. Instead, if the network was broken into a series of smaller networks connected through secure gateways or routers, a security problem in one of these smaller network can be better contained without having it affect the other networks. A crucial element of such an arrangement is that no workstation on one network should trust a workstation on

another network. As a further precaution, users who have accounts on more than one network should use different passwords for each.

Continuing with our previous analogy to the physical world, another way to limit the possibility of a fire would be to limit access to the building and to control the type of activities each individual granted access could perform. The ability to smoke in the building, for example, could be denied to all thus lessening the chance that a fire might start. The computer version of such policies is to limit individuals who can obtain entrance to a network, and to further restrict what type of activities these individuals can perform. One simple approach to perform this filtering is to force each host to individually determine who and what it will allow to access its services. An example of such an approach is *TCP Wrapper* [13.6], a tool designed at the Eindhoven University of Technology in The Netherlands and used throughout the UNIX community.

TCP Wrapper works by specifically identifying which network traffic to allow, and which to deny. It uses two files to describe the allowable accesses: */etc/hosts.allow* and */etc/hosts.deny*. Access can be controlled on a host or services basis, or a combination of both. For example, services such as *telnet*, *ftp*, and *rlogin* could be allowed while others such as *finger* could be blocked. In addition, specific hosts could either be blocked from certain types of access or granted certain access privileges. A drawback to this approach is that it requires that the system administrator to configure each system. A mistake in one of the files could allow an intruder access to one of the machines in the network which might then lead to further penetration of the network. A different approach is to place the filtering responsibilities for the network in a limited number of machines. This is the approach taken by most computer network firewalls.

13.2 Network Firewalls

The purpose of a network firewall is to provide a shell around the network which will protect the systems connected to the network from various threats. The types of threats a firewall can protect against include:

- *Unauthorized access to network resources* - an intruder may break into a host on the network and gain unauthorized access to files.
- *Denial of service* - an individual from outside of the network could, for example, send thousands of mail messages to a host on the net in an attempt to fill available disk space or load the network links.

- **Masquerading** - electronic mail appearing to have originated from one individual could have been forged by another with the intent to embarrass or cause harm.

A firewall can reduce risks to network systems by filtering out inherently insecure network services. Network File System (NFS) services, for example, could be prevented from being used from outside of a network by blocking all NFS traffic to or from the network. This protects the individual hosts while still allowing the service, which is useful in a LAN environment, on the internal network. One way to avoid the problems associated with network computing would be to completely disconnect an organization's internal network from any other external system. This, of course, is not the preferred method. Instead what is needed is a way to filter access to the network while still allowing users access to the 'outside world.' A typical network firewall can be depicted as shown in [Figure 13.1](#).

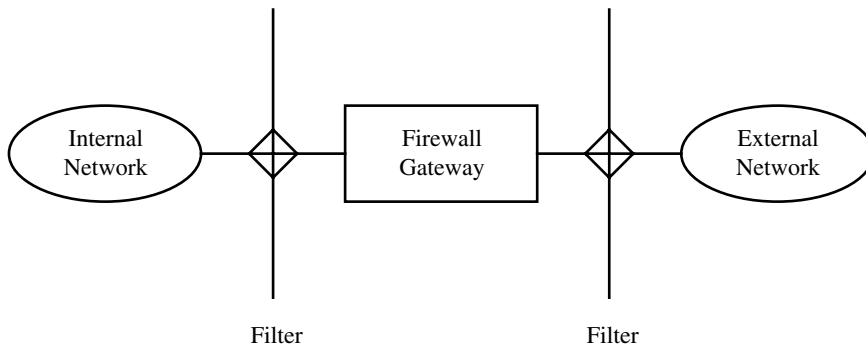


Figure 13.1. Typical network firewall configuration [13.1].

In this configuration, the internal network is separated from external networks by a firewall gateway. A gateway is normally used to perform relay services between two networks. In the case of a firewall gateway, it also provides a filtering service which limits the types of information that can be passed to or from hosts located on the internal network. There are three basic techniques used for firewalls: **packet filtering**, **circuit gateways**, and **application gateways** [13.1]. Often, more than one of these is used to provide the complete firewall service.

13.2.1 Packet Filtering Gateways

Packet filtering can provide an easy and cheap way to implement a basic level of filtering for a network. Most implementations involve a router used to connect two networks together. The router often is delivered with software which can enable it to discard packets based on the source or destination ports or addresses. The configuration of a typical router used as a firewall matches [Figure 13.1](#). In this example, the router replaces the gateway. The actual filtering can be applied to incoming packets, outgoing packets, or both. In order to perform the filtering, the router needs to know which packets to accept and which to reject. This information is contained in a file which defines a set of filtering rules. This rule set should include information about which sources and destinations to allow packets to arrive from or be sent to. An example of a set of rules is depicted in [Figure 13.2](#).

operation	source	port	destination	port	type
discard	bad.host	*	*	*	*
allow	our.host	25	*	*	*
discard	128.236.*.*	>1023	our.host	>1023	tcp

Figure 13.2. Sample packet filtering rules [13.1].

In this example, the rules instruct the router to discard all packets that originated from *bad.host*. It also instructs the router to allow all mail messages originating from *our.host* to any destination. The last line instructs the router to discard all *tcp* packets originating from a port greater than 1023 on any system from the 128.236 network which are destined for a port greater than 1023 on our system. An interesting question at this point is what happens to all of the other traffic which has not been described by the current set of rules? Should the router allow anything that it has not been specifically instructed to discard or should it discard everything that it has not specifically been told to allow? Obviously the safer of the two from a security standpoint is to discard all packets not identified as allowable but this may be too restrictive for some sites. The rules also have to be designed so that a mixture of *accepts*, *discards*, and *allows* can exist for a particular site and so that whole classes of network traffic (or services) can be denied. Consider the example depicted in [Figure 13.3](#).

operation	source	port	destination	port	Type
discard	bad.host	*	*	*	*
allow	our.host	25	*	*	*
accept	bad.host	25	our.host	25	*
discard	*	*	our.host	21	*

Figure 13.3. Additional packet filtering rules [13.1]

In this case, all traffic from *bad.host* except electronic mail (packets with a source and destination of port 25) is discarded. In addition, all FTP traffic (packets with a destination port of 21) is discarded without regard to its originating source. Administrators often deny whole classes of traffic because the class has proven to be too dangerous from a security standpoint. Examples of particularly dangerous classes of traffic include *tftp*, *sunrpc*, *rlogin*, and *rexec* [13.2].

A couple of slightly more complex, yet very common, configurations are depicted in Figures 13.4 and 13.5. In Figure 13.4, which is referred to as a Screened Host Gateway, the router is configured such that the Bastion Host is the only computer system on the internal network that can be accessed from an external network. An exposed host that is designed to be a critical strong point in the security of a network or group of networks is often referred to as a bastion host [13.1, 13.3] (a bastion, historically, is a particularly strong part of a fortification or castle). Since the bastion host is the only system accessible to outsiders, it is imperative that it is protected against intrusions. Should the bastion host become compromised, the rest of the internal network is in danger.

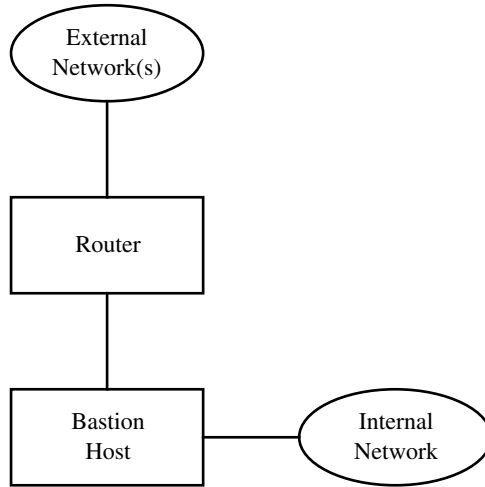


Figure 13.4. A Screened Host Gateway [13.3].

Often, the router is called upon to perform several functions. In [Figure 13.5](#) a single router is used to not only serve as the network firewall but to also route traffic between several internal networks. The first internal net, Net 1, consists of only a gateway which is used for communication with the external world. All traffic destined for an external host must be sent through the gateway. Likewise, traffic from an external host destined for a host on one of the internal nets must pass through the gateway. This arrangement allows for an additional level of security since the gateway (or bastion host) is not part of the secure internal networks. The network inhabited by a gateway acting as a bastion host is often referred to as the demilitarized zone (DMZ) because of the security danger that exists for this system to be exposed to external threats from other networks.

The router will allow any traffic to pass between the non-DMZ internal networks (Net 2 and Net 3). External communication between any host on either Net 2 or Net 3 and any host on an external network, however, must pass through Net 1 and the Gateway. Communication between the Gateway on Net 1 and machines on Nets 2 and 3 can also be restricted, thus adding an additional level of security.

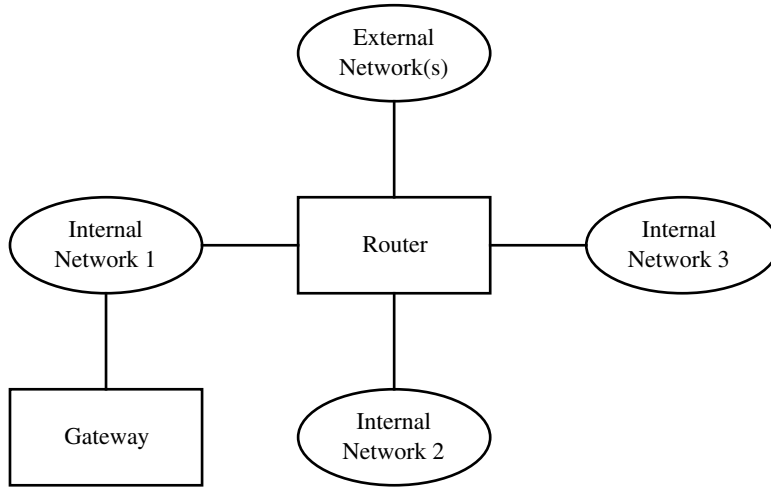


Figure 13.5. Firewall router with two internal subnets.

Sometimes, in order to speed processing, filtering is only accomplished on either input or output packets but not both. In fact, some routers are designed to only filter based on the destination port [13.1]. If only output filtering were to be performed on the network configuration of [Figure 13.5](#) a problem could arise if an individual from an external host attempted to forge a packet address. This individual on an external net, for example, could send a packet to the router that claimed it was from one of the other internal networks. If only output filtering was performed, this sort of attack could not be prevented. If only output filtering is to be performed, the configuration of [Figure 13.4](#) can be slightly modified to handle address forgery. This new configuration uses one router to handle routing functions for the internal networks and another router as the interface to the external world. Thus an individual who sent a packet from an external address with an address forged to appear as if it came from one of the internal networks would be caught by the second router and not allowed to pass to the internal router and networks. This special configuration is shown in [Figure 13.6](#) and is referred to as a Screened Subnet [13.3].

One final comment needs to be made about the need to filter routing information as well as other traffic. In the configuration depicted in [Figure 13.5](#), Nets 2 and 3 were not allowed to send traffic directly to any external networks. Another advantage of this is that the paths to hosts on Nets 2 and 3 can be hidden from external networks since the Gateway can be used to correctly direct traffic in the internal nets. This effectively hides the actual paths for machines on Nets 2 and 3. The router, however, needs to keep track of paths to hosts on these networks but it should not send routing

information about any hosts on Nets 2 or 3 to external networks. This information needs to be filtered.

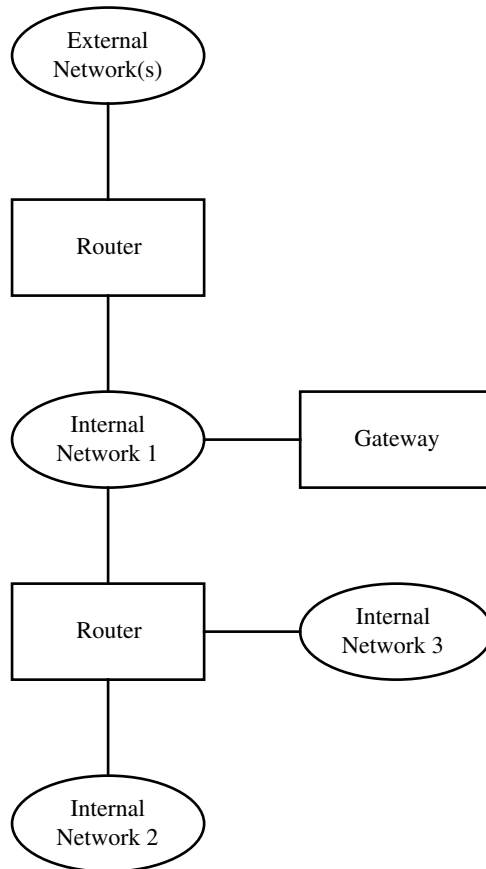


Figure 13.6. Firewall for routers with output filtering [13.1].

13.2.2 Circuit Level Gateways

Circuit-level gateways simply relay network traffic between two hosts connected via a network's virtual circuit [13.1]. An internal user, for example, might connect to a port on the gateway which would in turn connect to another port on a host residing in an external network. The gateway simply copies the bytes from one port to the other. Normally the gateway will relay the data without examining it but often may keep a

log of the amount of data relayed and its intended destination. In some instances the relay connection (forming a 'circuit') will take place automatically for specific network functions. At other times, however, the gateway will need to be told the desired destination port. When the connection request is made of the gateway, the decision as to whether this is an allowable transaction is made. If it is allowed, the circuit connection is created and the data relaying begins. If, on the other hand, the requested connection is deemed inappropriate then the connection is not made and an error message can be returned (or the request simply discarded). Although normally thought of as a relay for TCP traffic, circuit-level gateways can also be used for some UDP applications where a virtual circuit is assumed.

13.2.3 Application Level Gateways

Application level gateways (also referred to as *proxy gateways*) usually operate at a user level rather than the lower protocol level common to the other firewall techniques. Instead of routing general purpose traffic, application-level gateways are used to forward service-specific traffic. A common example of this is a mail gateway. Indeed, the use of gateways to direct electronic mail traffic is so common that the Domain Name Service (DNS) has a special feature called MX records (mail exchanger) for their support [13.1]. The use of such gateways not only is valuable from a security standpoint by isolating internal networks, but also aids in the everyday use of this service. Users, for example, can access email from different machines while still keeping the same email address. In addition, just like in circuit-level gateways, application-level gateways provide a centralized point for the logging of activities. The obvious disadvantage to application-level gateways is the need for a special program to handle each desired application. Since this requires quite a bit of work, typically only those services deemed to be extremely important will be supported.

The basic configurations are similar to those found in firewalls utilizing packet filtering routers and bastion hosts. The most basic design for an application-level gateway firewall is similar to [Figure 13.4](#). This configuration is more secure than a filtering router alone, however, since it will only allow certain, very specific, traffic to get through. This is probably the most common configuration for application-level firewalls.

The other common configuration is shown in [Figure 13.7](#). Note that this design is similar to [Figure 13.6](#) with the addition of a second gateway attached to the internal net. In this firewall design the inside router (the one that connects all internal networks as well as the DMZ) serves as a *choke*. The purpose of the choke is twofold: first to block all internal traffic destined for external networks unless addressed to the internal gateway, and second to block all external traffic destined for the internal net unless it is

also addressed to the internal gateway. This design (when all routers and gateways are properly configured) results in a very secure firewall.

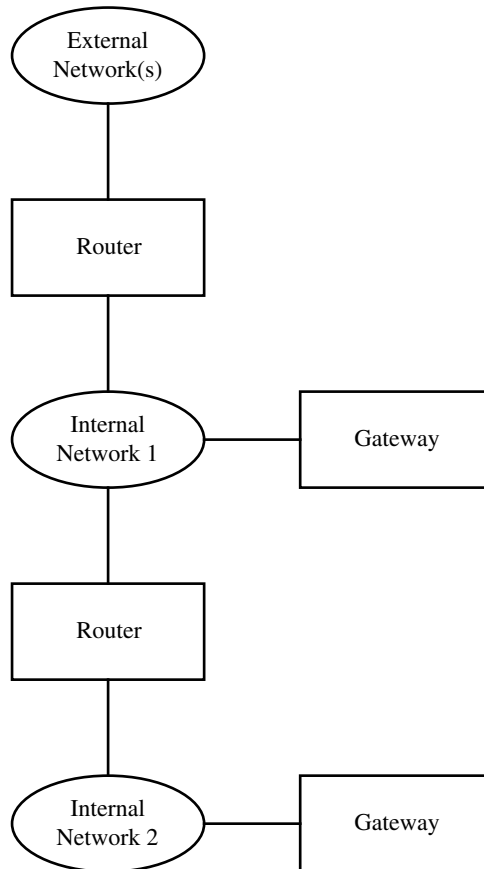


Figure 13.7. A common application-level gateway design [13.1].

13.3 Firewall Costs and Effectiveness

It should be obvious that with the additional software and hardware that is required to implement a firewall comes an associated cost both in terms of performance and dollars. There is also the additional cost associated with the not so insignificant amount of time required for the administration of the firewall. System performance

may be affected since certain services may not be available and others are forced to be channeled through a single machine. What happens, for example, when the machine acting as an organization's internal gateway as shown in [Figure 13.7](#) malfunctions? The internal net and its hosts are functioning but they can't communicate with external nets because all traffic is relayed through the gateway. Of course, all of these costs must be weighed against the possible costs associated with a breach in security. How much will it cost in terms of lost business or the time required to "clean up" a system should an intruder gain access to the organization's computing assets? Firewalls, however, may in fact prove to be less expensive overall as they may concentrate the majority of an organization's security service in the firewall system(s). They may also provide additional benefits in terms of being able to provide valuable statistics on network usage. Additional benefits such as this can mitigate some of the extra costs associated with firewalls.

An additional non-monetary cost associated with firewalls is the loss of a certain amount of freedom. Anytime restrictions are placed on the provided services a certain segment of users will feel that their ability to perform assigned tasks has been impeded, since access through the firewall is more difficult. For the most part, however, the obvious need for a level of security to protect an organization's (and the individuals in the organization) computing assets results in an acceptance of these limitations.

While firewalls are an important tool to be used in defending a network, they are by no means perfect. One of the largest limitations of firewalls is that they are only as secure as the service whose traffic they allow through. In other words, if the application program/protocol that an organization is running has errors in it which might result in a security hole, the firewall will not be able to catch it. The firewall's job was to simply restrict the traffic that got through. It is not the responsibility of the firewall to monitor the operation and semantics of the applications and services themselves.

Another way to circumvent the protections provided by a firewall is to *tunnel* through them. Tunneling involves the encapsulation of a message of one protocol inside another [13.1]. This encapsulated message then traverses the target networks, traveling through any firewalls that are in place, until it reaches the destination host. At the destination host, the encapsulation is stripped off and the message in its original protocol is inserted into the network. It is obviously important that the outer protocol be one which the firewalls are designed to let through.

13.4 Sample Security Packages

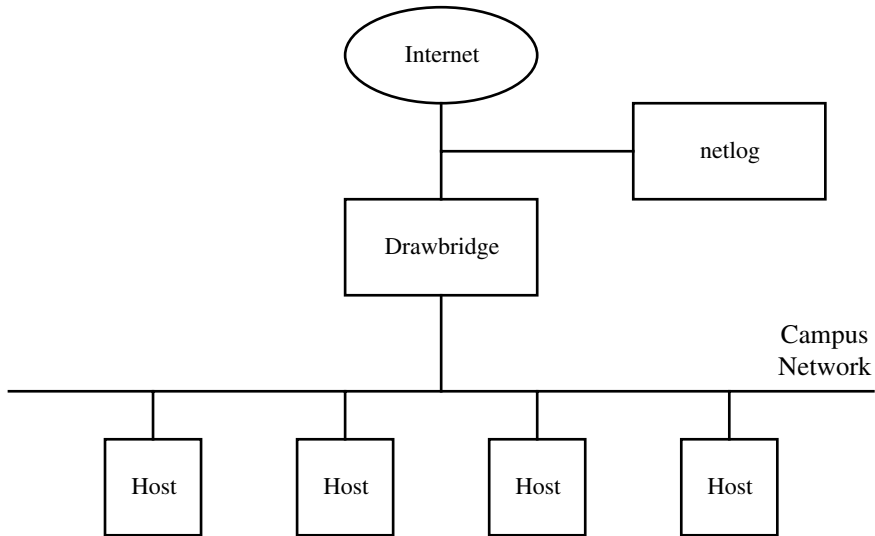
In August of 1992, the Texas A&M University Supercomputer Center was notified that a computer owned by the university was being used as a platform to

launch attacks against other computer systems around the world. As is discussed in Chapter 17, *Case Studies*, this led to the development of three security packages: **drawbridge** – a filtering tool, **tiger** – a set of programs which can be used to check machines for security problems, and **netlog** – a series of network intrusion detection programs [13.5].

The first program in the Texas A&M University (TAMU) security packages is **Drawbridge**. This program is designed to filter all internet packets transmitted to or from TAMU. It allows access to be controlled on a machine by machine as well as port by port basis. **Drawbridge** is a simple tool, more akin to the use of routers for filtering rather than the more complicated firewalls discussed in this chapter. Recognizing that it was only the first line of defense, **netlog** was developed to monitor the incoming network traffic in order to determine if an attempt was being made to circumvent the filter (**drawbridge**). **Netlog**, which actually consists of a set of monitoring tools, checks the incoming traffic for unusual connections (attempts to connect to **tftp** for example), unusual patterns of connections (based on historical data gathered on ‘normal’ network usage at TAMU), and a wide range of detailed intrusion signatures (including such things as attempts to login using system accounts or to gain **root** access). The third element of the TAMU security package is **tiger**, a series of scripts (and thus generally referred to as the **tiger scripts**) used to check a computer system for signs that an intrusion has occurred, as well as to determine if certain security related configuration items are set correctly. The scripts check target machines for such things as:

- Accounts without passwords.
- Directories often used by intruders for signs of unexpected files.
- World readable home directories.
- Digital signatures of system binary files.
- .rhost files with ‘+’ entries.

The various components of the TAMU security package are shown in [Figure 13.8](#). Additional information on the security package, including details on how to obtain copies, can be found in [13.5].



Tiger Scripts
Employed on all Hosts

Figure 13.8. Texas A&M University Security Package [13.5].

Another security package available to help secure a computer network is the Trusted Information Systems (TIS) Firewall Toolkit—the foundation for numerous commercially available firewalls including that depicted in [Figure 13.4](#). This toolkit is available at no charge from TIS over the Internet. The TIS Firewall Toolkit is designed for use on UNIX systems using TCP/IP. It can be used to build Screened host-based firewalls utilizing a bastion host to enforce security as was depicted in [Figure 13.4](#). The security applications provided by the toolkit include:

- An electronic mail gateway
- An FTP application gateway
- An audit log of significant security events
- A TELNET application gateway
- A ‘plugboard’ gateway for other TCP connections (such as network news)

The toolkit requires installation by a systems manager experienced in UNIX systems. Additional information on the toolkit including details on how to obtain copies can be found in [13.4].

13.5 Summary

The complexity involved in securing computer systems connected to a network increases with the addition of every new machine. Since systems often are designed to implicitly trust other systems they are ‘close’ too, a single insecure system in a network can place the entire network in jeopardy. A stop-gap measure, used to help protect the machines connected to a network from individuals outside of the networked environment, is the introduction of firewalls to the environment.

Firewalls can be implemented at different levels of a network. Packet filtering gateways can be configured in a variety of ways. Most utilize a router to perform some initial filtering for the networks it protects. In addition, many utilize a special *bastion host* which is often the only host on a network that is accessible to the outside environment. This simplifies the security problem since now only a single machine must be trusted to protect against external intruders. Other firewalls have been designed for application at the Circuit and Application levels.

Many routers are shipped with simple filtering routines which can be used to build a first line of defense against intruders. Several more extensive packages are available free of charge to organizations in need of firewall protections [13.4, 13.5]. In addition, several vendors have developed their own firewall packages for either internal use or for commercial sales. While firewalls can add an additional level of security for a network, they come at a cost. Additional software and often dedicated hardware is required to install firewalls. They also will restrict what services will be permitted to pass outside of the internal network which may be viewed by some as a costly restriction. Firewalls are also not foolproof as unrestricted modem access to an internal network could entirely negate the purpose of the firewall.

13.6 Projects

- 13.1 What are the problems and tradeoffs associated with having a packet filtering router allow all traffic not specifically identified to be discarded versus having the router discard all traffic not specifically identified as being allowed?
- 13.2 Special precautions need to be taken for firewalls which consist of routers which only perform output filtering. Are there any special problems associated with firewalls which only filter incoming traffic? Why?

- 13.3 Explain why the term “proxy gateway” is descriptive of the nature of application-level gateways.
- 13.4 How can the firewall monitoring and dummy user accounts be used together to detect intrusive activity on networks?
- 13.5 What challenges and problems does the growth of information protocols such as *gopher* and the *World Wide Web* present to firewalls?
- 13.6 What problems does the Mbone (the Internet’s multicast backbone) present to firewalls?

13.7 References

- 13.1 Cheswick, William R. and Bellovin, Steven M., Firewalls and Internet Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1994.
- 13.2 Garfinkel, Simson and Spafford, Gene, Practical UNIX Security, O’Reilly & Associates, Inc., Sebastopol, California, 1992.
- 13.3 Ranum, Marcus, “Thinking About Firewalls”, Trusted information Systems, Inc., available for anonymous FTP from *ftp.dsi.unimi.it:pub/security/docs/firewall/fwalls.ps.gz*, January 1995.
- 13.4 Ranum, Marcus and Avolio, Frederick, “A Toolkit and Methods for Internet Firewalls”, Trusted Information Systems, Inc., available for *anonymous ftp* from *ftp.dsi.unimi.it*, *pub/security/docs/firewall/usenix-paper.ps.gz*, January 1995.
- 13.5 Safford, David; Schales, Douglas and Hess, David, “The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment”, *Proceedings of the Fourth USENIX Security Symposium*, Santa Clara, California, October 1993, pp. 91-118, also available via *anonymous ftp* from *net.tamu.edu*, */pub/security/TAMU*.
- 13.6 Venema, Wietse, “TCP WRAPPER: Network Monitoring, Access Control, and Booby Traps.”, *Proceedings of the Third Usenix UNIX Security Symposium*, Baltimore, Maryland, September 1992, pp. 85-92.

13.8 Extended Bibliography

- 13.7 Avolio, Frederick M. and Ranum, Marcus J., “A Network Perimeter with Secure External Access”, Trusted Information Systems, Inc., Glenwood, MD, available for *anonymous ftp* from *ftp.dsi.unimi.it*, *pub/security/docs/firewall/isoc94.ps*, January 1995.
- 13.8 Cheswick, Bill, “The Design of a Secure Internet Gateway”, AT&T Bell Laboratories, Murray Hill, New Jersey, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/SIG.ps.gz*, January 1995.
- 13.9 Digital Equipment Corporation, “Screening External Access Link (SEAL) Introductory Guide”, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/SEAL.ps.gz*, January 1995.
- 13.10 Livermore Software Laboratories, Inc., “PORTUS Announcement Letter”, Houston, Texas, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/lsl_i_portus.ps.gz*, December 1994.
- 13.11 Mogul, Jeffrey, “Simple and Flexible Datagram Access Controls for Unix-based Gateways”, Technical Report 89/4, Western Research Laboratory, Digital Equipment Corporation, Palo Alto, California, March 1989, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/screen.ps.gz*, January 1995.
- 13.12 Ranum, Marcus, “A Network Firewall”, *Proceedings fo the World Conference on System Administration and Security*, Washington, DC, July 1992, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/f_dec.ps.gz*, January 1995.
- 13.13 Raptor Systems Inc., “Eagle Network Security Management System: User’s Guide Version 2.2”, Wilmington, Delaware, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/Eagle.ps.gz*, January 1995.
- 13.14 Reid, Brian K., “The DECWRL Mail Gateway”, Digital Equipment Corporation, available for *anonymous ftp* from *ftp.dsi.unimi.it*: *pub/security/docs/firewall/DecMailGateway.ps.gz*, January 1995.

- 13.15 Siyan, Karanjit and Hare, Chris, Internet Firewalls and Network Security, New Riders Publishing, Indianapolis, Indiana, 1995.
- 13.16 Skoudis, Edward, “Fire in the Hole”, *Information Security*, Vol. 1, No. 8, July 1998, pp. 16-23.
- 13.17 Treese, G. Winfield and Wolman, Alec, “X Through the Firewall, and Other Application Relays”, Technical Report CRL 93/10, Cambridge Research Laboratory, Digital Equipment Corporation, Cambridge, **Massachusetts**, May 1993, available for *anonymous ftp* from **ftp.dsi.unimi.it**: pub/security/docs/firewall/ CRL_firewall.ps.gz, January 1995.
- 13.18 Wack, John P., and Carnahan, Lisa J., “Keeping Your Site Comfortably Secure: An Introduction to Internet Firewalls”, NIST Special Publication 800-10, U.S. Department of Commerce, National Institute of Standards and Technology, available on the internet at csrc.ncsl.nist.gov/nistpubs/800-10/, October 1998.

14

CRYPTOGRAPHY

Cryptography comes from the Greek words *kryptos* meaning hidden or secret and *graphos* meaning writing. Once the concern of diplomats and military officers, today cryptography is much more widespread. From the automated teller machines used by banks, to private subscription television transmissions, cryptography is part of everyone's life. What was once a topic discussed only behind locked doors of organizations such as the National Security Agency now is a major academic topic, an active area of research, and a common aspect of business and industry.

The discipline relating to the use and development of techniques to encrypt and decrypt messages is called ***cryptography***. An attempt to break a specific cryptographic technique is called ***cryptanalysis***. Usually it is assumed that an individual only has a copy of an encrypted message when trying to break a specific technique in order to read the message. The process is often made easier if the cryptanalyst can obtain the encrypted version of some known message. The field which covers both cryptography and cryptanalysis is known as ***cryptology***.

The process of ***encryption*** entails taking a message (often referred to as ***plaintext*** or ***cleartext***) and changing it to hide the original meaning from everybody but the intended recipient(s). ***Decryption*** is the process that takes the encrypted message (now referred to as ***ciphertext***) and restores it to the original message.

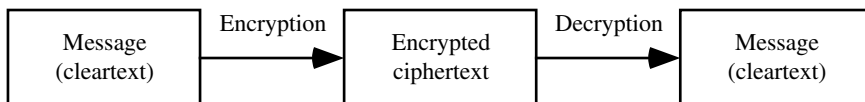


Figure 14.1. The Encryption and Decryption Processes.

This process of changing plaintext to ciphertext and back again requires that a pair of transformations takes place. These transformations use mathematical functions which incorporate an additional piece of data, known as the *key*, to perform the required transformations. The key is kept secret so that only the intended recipient(s) can decrypt the message. These transformations can be represented as follows:

$$Ciphertext = Encrypt_{[key]}(Plaintext) \quad (14.1)$$

$$Plaintext = Decrypt_{[key]}(Ciphertext) \quad (14.2)$$

The desire to keep information and messages hidden from others is not a new phenomenon which has appeared with the advent of computers. Governments, the military, and businesses have used cryptography for centuries to hide their secrets from potential or actual adversaries or competitors. One of the first encryption methods was a simple substitution technique which disguised each individual character.

14.1 Substitution Ciphers

Substitution Ciphers are based on the principle of replacing each character with another character in order to hide the actual meaning of the message. Substitution by itself is not new nor is it always used to hide the meaning of a message. A well known example of substitution is Morse code which was created not to keep a message secret but rather to facilitate transmission of messages via various media. Another substitution is the Braille alphabet which is used by those who cannot see normal printed characters.

14.1.1 Caesar Cipher

The famous Caesar Cipher was a simple substitution cipher which replaced each letter with one three to the right in alphabetical order. Using this technique, A would become D, B would become E, and so forth. The last three characters of the alphabet would wrap so that X becomes A, Y becomes B, and Z would become C. Using this technique, the word “computer” would become

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

Cleartext \Rightarrow	C O M P U T E R
Ciphertext \Rightarrow	F R P S X W H U

The decryption process is a simple matter of replacing all characters in the ciphertext with the character three to the left (with the characters A, B, and C now wrapping around to X, Y, and Z respectively).

14.1.2 ROT13

A common simple cipher which can be found on some UNIX systems is ROT13. Like the Caesar Cipher, ROT13 simply replaces each plaintext character with its equivalent this time 13 spaces to the right instead of just the three spaces used by the Caesar Cipher. This technique has the advantage that the same transformation function can be used for both encryption and decryption since, with 26 characters in the alphabet, shifting 13 characters to the right is equivalent to shifting 13 characters to the left (with appropriate wrap-around). The C code for ROT13 is as follows [14.10]:

```
main ()
{
    int chr;

    while (( chr = getchar()) != EOF)
    {
        if (chr >= 97 && chr <= 109) chr=chr+13;
        else if (chr >= 110 && chr <= 122 ) chr=chr-13;
        else if (chr >= 65 && chr <= 77 ) chr=chr+13;
        else if (chr >= 78 && chr <= 90 ) chr=chr-13;
        putchar(chr);
    } /* end of while */
} /* end of ROT13 */
```

14.1.3 Substitution Cipher Variations

An alternative method to the previous, very simple substitution techniques, is to create a random sequence of letters and use this sequence to guide the transformation. For example, if the following sequence was chosen:

zxcvbnmasdfghjklqwertyuiop

the ciphertext could be obtained by lining the regular sequence above the key and making the appropriate substitution, as follows:

ABCDEFGHIJKLMNOPQRSTUVWXYZ
zxcvbnmasdfghjklqwertyuiop

Using this method the message:

Now is the time for all good men

becomes:

Jku se rab rshb nkz zgg mkkv hbj

This message can be made a bit more unreadable by removing the spaces between the words and then grouping the message into blocks of characters as follows:

Jkuse rabrs hbnkw zggmk kvhbj

The weakness in all substitution techniques is that some letters are more frequently used than others. This allows an individual, by simply calculating the number of occurrences for each character in the ciphertext, to determine the substitutions that were made. In English text, the expected frequency of specific letters for a 1,000 character document can be seen in [Table 14.1](#). From this table we can see that the most commonly occurring letter in English is E. In a 1,000 character document we would thus expect to find approximately 125 E's.

Simply substituting another character merely changes the frequency for specific characters but maintains the relative frequency that characters are used. Another problem with substitution techniques is that certain pairs of letters are more often used than others. This allows the cryptanalyst to search for commonly found patterns. This is especially true for characters that are repeated in a word. In our example above, the doubles 'gg' and 'kk' both appear in the ciphertext. Only certain letters are commonly found repeated in words (e.g., 'oo', 'mm', and 'll'). This characteristic of the language further helps the analyst to break the ciphertext.

Table 14.1. Expected number of occurrences for each letter of the alphabet in a 1000 word English text document [14.11]

Letter	Occurrences	Letter	Occurrences	Letter	Occurrences
E	125	L	36	W	15
T	90	H	34	V	14
R	83	C	33	B	12
N	75	F	29	K	5
I	74	U	28	X	4
O	73	M	26	Q	4
A	71	P	25	J	2
S	58	Y	22	Z	2
D	40	G	20		

14.1.4 Vigenere Ciphers

The goal of a substitution cipher should be to break up the natural frequency of occurrence for letters and groups of letters. One way to accomplish this is to change the substitution key during the message. For example, if we chose the numeric key:

3 5 7 9 11

we would shift the first letter three characters, the second five, the third seven, and so forth with the sixth letter starting the sequence over. This yields the following ciphertext for our earlier message:

Now is the time for all good men
Qtd rd wml ctpj mxc dqs pzri tny

This method starts to eliminate some of the problems associated with simple substitution ciphers and uses what is known as a Vigenere Table. A full Vigenere Table is shown in [Table 14.2](#).

Table 14.2. Vignere Table

0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
2	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
3	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
4	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
5	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
6	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
7	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
8	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
9	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
10	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
11	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
12	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
13	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
14	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
15	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
16	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
17	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
18	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
19	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
20	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
21	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
22	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
23	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
24	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
25	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U

When using a Vignere table, the key may be a word which both parties know and which indicates the row of the table to use. If, for example, the word “computer” were chosen for the key, the row of the table to use could be selected several ways. One method is to repeatedly write the key over the plaintext. The specific row to use for the substitution is the one that starts with that letter of the alphabet. This will yield the following ciphertext for our previous example:

COM PU TER COMP UTE RCO MPUT ERC	(repeated key)
Now is the time for all good men	(plaintext)
Pci xm mlv vwyt zhv rnz sdiw qvp	(ciphertext)

Another method which can be used is to base the row selected on the sorted order of the letters in the key. In our example the letter C is the first letter (alphabetically), E is the second, M the third, and so forth. Our earlier example would then be encrypted as follows:

C O M P U T E R	(key)
1 4 3 5 8 7 2 6	(sorted order)
143 58 726 1435 872 614 3587 261	(repeated key)
Now is the time for all good men	(plaintext)
Osz na ajk umpj nvt gmp jtwk oko	(ciphertext)

Notice that these two methods resulted in different ciphertexts even though the same key was used. Obviously, both sides of the encryption process must use the same method for a transmitted message to be correctly decrypted.

Using a word such as “computer” as a key, instead of a random selection of numbers, makes it easier to remember. While the methods shown have made it more difficult to determine the correct substitutions, they are still not perfect. Continuing with our plaintext example we can obtain the following additional ciphertext:

43 5872 61 435 872 61 43587 2614358726
to come to the aid of their countrymen
xr hwtg zp xkj ipf ug xkjgy euvrwtgt

If we were to examine the ciphertext we would first notice that the frequency in which letters occur does not match the expected frequency. We therefore know that a simple substitution has not occurred. If we were to further examine the ciphertext, however, we would notice that several groups of letters occur in the same sequence on more than one occasion. The pair ‘tg’ appears three times while the pair ‘mp’ appears twice as does the pairs ‘gt’ and the triplet ‘xkj’. The pair ‘mp’ occurs seven characters apart, the pair ‘tg’ occurs fifteen and twenty-four characters apart, ‘gt’ two characters, and ‘xkj’ eight. This gives us a clue as to what length the key might be. Three of the pairs have two as a factor and two have eight as a factor. If we were to then group the letters into groups of two (assuming the key is two letters long) and then eight (which assumes an eight character key) we get:

O s	O s z n a a j k
z n	u m p j n v t g
a a	m p j t w k o k
j k	o x r h w t g z
u m	p x k j i p f u
p j	g x k j q y e u
.	
.	

We could now check to see if the resulting groupings yielded the correct character occurrence frequency for each of the columns. If we found one that did, it might indicate the length of the key and allow us to break the cryptosystem. It should be obvious that the longer the captured ciphertext is the easier that it will be to break. It should also be obvious that the closer the key size is to the size of the plaintext, the harder it will be to break.

14.1.5 One-Time Pads

Several techniques have been described to make breaking a cryptosystem difficult but there is one method, invented in the early part of this century, that results in an unbreakable code. This method consists of what is known as one-time pads. Each pad contains several pages, each consisting of a large number of characters in random order. Each page of the pad is used only once to encrypt a message and is then discarded. Messages longer than the length of a single page use multiple pages so that the sequence is not repeated. The intended recipient of the ciphertext must have an identical pad as the sender. Since any key sequence from the pad is equally as likely, there is no way, given the ciphertext alone, to determine what the original plaintext message was. Consider the following plaintext message:

One if by land

If the key consisted of the letter sequence:

thxysiesgdw

then, by using the key to determine which line of a Vigenere table to use, the substituted ciphertext we would obtain is:

hub gx jc dgqz

For the cryptanalyst who is trying to decipher this message, if the key:

oynysielcqh

was guessed, the ciphertext would be decrypted as:

Two if by seas

which is just as likely, from the cryptanalyst's perspective, as the original plaintext. In fact, any message of the same length is equally as likely with no way of determining which is correct.

Since the one-time pad provides such secure communication it might at first appear as the solution to all cryptographic problems. In actuality, however, the one-time pad suffers from one very big disadvantage, the pad itself. Pads must be distributed in a secure manner to all parties involved. This same problem exists for all symmetric cryptosystems and is known as the **key distribution** problem. In addition, the pads must contain random sequences longer than the plaintext message. With the increasing size and amount of encrypted communication seen today, one-time pads do not offer a practical solution.

14.2 Transposition Ciphers

Transposition ciphers differ from substitution ciphers in that they do not change the characters themselves but rather the order in which they appear in the message. A simple example, using our previous message, is as follows:

won si eht emit rof lla doog nem

In this extremely simple example, the order of the characters have been reversed in each word while each character maintains its original identity. This means that the frequency of occurrence for each letter will not change and provides a clue that a transposition cipher is being used. While the above example is easily read, using other methods to transpose the character positions yields significantly more difficult ciphertext. The plaintext can be arranged, for example, in two vertical columns, then taken in horizontal pairs and placed into groups of five characters yielding:

N	o	
o	r	
w	a	
i	l	
s	l	
t	g	Noorw ails1 tghoe otdim meenf
h	o	
e	o	
t	d	
i	m	
m	e	
e	n	
f		

Various numbers of columns can be used for the transposition process. Breaking this method of encryption consists of repeatedly using a different number of columns. Another transposition method consists of using various geometric patterns to transcribe the message. If we group the text into a square, we obtain:

```
N o w i s  
t h e t i  
m e f o r  
a l l g o  
o d m e n
```

We can then transcribe the message by taking the characters in any of several different patterns. We could, for example, start at the top and follow a spiral pattern in a counter-clockwise direction yielding:

```
N t m a o d m e n o r i s i w o h e l l g o t e f
```

Another approach would be to transcribe the message in a diagonal pattern. Starting at the top right and proceeding down and to the left would yield:

```
s i i w t r o e o o N h f g n t e l e m l m a d o
```

Should the plaintext message not be of the correct size, extra characters can be added to reach the required number for the desired block.

14.3 Encrypting Digital Communication

When considering the seven-layer Open Systems Interface (OSI) network model, encryption can take place at any of the layers. In practice there are two different approaches that are commonly seen. **Link Encryption**, which takes place at the physical layer and **End-to-End Encryption**, which occurs between the network layer and the transport layer (or any of the other higher level layers).

At the physical layer, encryption devices ('black boxes') can be easily attached where the physical media exits each node. Decryption takes place at the other end of the media as it enters a node. This effectively encrypts all transmissions across the media. One particular advantage to this form of encryption is that it protects against **traffic-flow analysis**. Since everything that is sent along the media is encrypted, an individual who taps into the media would not even be able to determine the destination of the messages. A drawback to link encryption, however, is that it requires a large number of encryption devices and even one unencrypted link jeopardizes the

security of the whole network. In addition, each node must be protected since all traffic is decrypted at each node. All users of the nodes must be trusted because all traffic is decrypted at each node, allowing anybody to read all traffic transmitted across the media.

In end-to-end encryption the ciphertext remains encrypted as it travels through each node until it reaches its final destination. The destination is not encrypted so traffic-flow analysis can take place but the message remains encrypted at each node so the individual nodes need not be secure nor do all of the users need to be trusted individuals.

Another characteristic of digital encryption techniques is whether the encryption is done in a block or stream. **Stream ciphers** convert the plaintext one bit at a time. Stream ciphers are not normally implemented at the software but at the hardware level. **Block ciphers**, on the other hand, work on blocks of either plain or ciphertext. The largest difference between the two is in error propagation. An error in a stream cipher will corrupt a single bit while errors in block ciphers will corrupt an entire block. Historically blocks tend to be 64, or multiples of 64, bits in length.

A **symmetric** cryptosystem is one that uses the same key for both the encryption and decryption processes. If two individuals wish to communicate they obtain a data value to be used as the key (how they obtain the same value will be discussed later). This value is kept secret from all other individuals which allows the two parties to protect the contents of a message encrypted using this key from all others. The symmetric cryptosystem has three main problems:

- 1) If a third party determines the key, the messages sent between the two original parties would be compromised. In addition, the third party could send encrypted messages to fool the other parties.
- 2) The key must be distributed in a secret manner. In the past, this often has been accomplished by using couriers to distribute the keys.
- 3) A large number of keys are required to enable individual pairs of users to secretly communicate. The number of keys necessary is in fact $(n^2-n)/2$. Thus if there were 10 users only 45 keys would be needed, but if there were 100 users 4950 keys would be required, and if there were 10,000 users 49,995,000 keys would be necessary.

14.3.1 DES

The Data Encryption Standard (DES) has been in use worldwide as a standard for more than a decade. It arose during a period when several companies were producing

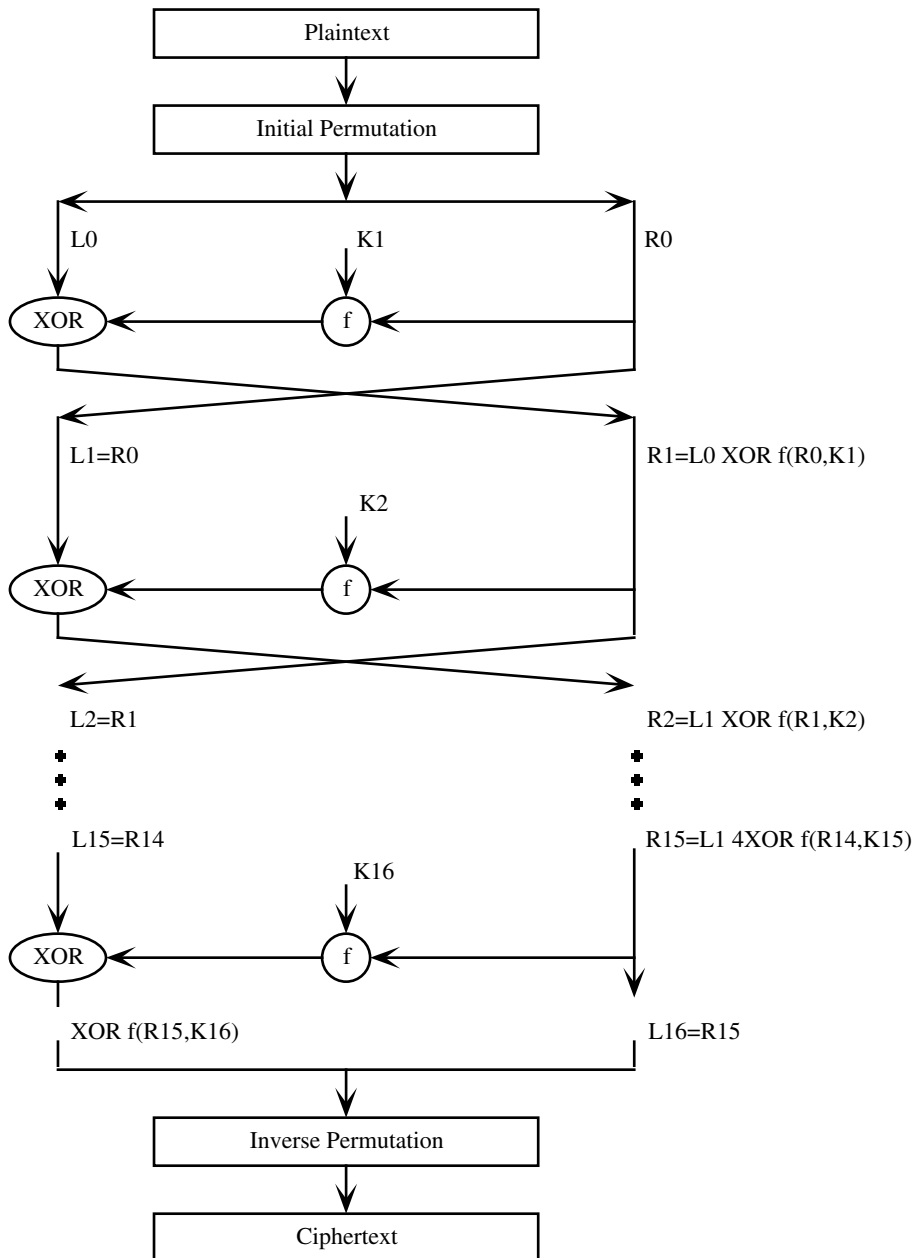


Figure 14.2. DES Encryption Computations [14.4].

cryptographic equipment which wasn't compatible and whose algorithms strength and security was unknown. The National Bureau of Standards (NBS, now called the National Institute of Standards and Technology or NIST) initiated an effort in 1972 and 1973 to develop a standard algorithm to protect digital data via a formal request for proposals. IBM answered this request with an algorithm based on an earlier one called LUCIFER (it is interesting to note that the original LUCIFER key was 112 bits long but after the National Security Agency was asked to help with the evaluation of the algorithm it was shortened to the current 56 bits). The resulting algorithm was officially adopted as a federal standard in 1976 and authorized for all federal government unclassified communications. The NBS issued an official description of the algorithm in the Federal Information Processing Standards Publication 46 (FIPS PUB 46) in 1977 [14.4].

The DES algorithm is designed to encrypt or decrypt blocks of data consisting of 64 bits using a 64-bit key. The same key is used for both the encryption and decryption processes. The block is first subject to an initial permutation (see [Figure 14.2](#)) as follows:

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

This means that the block after the initial permutation will have the 58th bit of the plaintext block as its first bit, the 50th bit as its second bit, and so forth. After the initial permutation, the block is then subjected to a series of key-dependent computations. Finally, the block permutes once more via the inverse of the original permutation.

Following the initial permutation, the block of data is split into two halves consisting of 32 bits each. These halves are combined with the key and processed through a function 'f' sixteen times. After the final iteration, the two halves are combined again and sent through the inverse permutation operation. The DES key consists of 64 bits, 56 of which are used by the algorithm with eight bits used for error detection. The function 'f' consists of four operations:

- 1) the key is shifted and 48 bits are selected
- 2) the right half of the data consisting of 32 bits is expanded to 48 and then an exclusive-or operation is performed on it and the 48 bit key
- 3) 32 bits of this new value are selected using what has been called the 'S-boxes'
- 4) a final permutation on these last 32 bits are performed [14.4, 14.10]

One benefit of this algorithm is that the same process can be used for both encryption and decryption (with the exception of the key which must be used in reverse order for decryption). The algorithm also has the advantage that it only uses logical operations resulting in a fast process whether implemented in hardware or software.

The S-Boxes play a crucial role in both the algorithm itself and the debate that surrounds its development. The cryptosystem must take the 48 bits that have been generated in the previous operation and select only 32 bits from it. Which 32 bits should be chosen and why? The answer to this is shrouded in some mystery since NSA has not allowed IBM to release why certain bits are chosen. Some speculate that this is because there may exist a quick way to decipher the message. As long as the reasons are not known, the cipher is reasonably secure from all; except NSA.

An ongoing debate has been exactly how secure is DES? There has been much suspicion that when NSA got involved, a 'trap door' was inserted so they could decrypt any DES encrypted messages (the mysterious 'S-Boxes'). A special U.S. Senate committee investigated this matter and, while the full transcript of their findings is classified, an unclassified summary cleared NSA of any improper manipulation of the algorithm. Debate also surrounds the issue of the DES key length. Some have speculated that there exist certain governments and organizations which may have built specialized DES breaking machines which utilize a brute-force attack. A longer key, some argue, would have made this option prohibitively expensive.

One known weakness in DES surrounds the selection of the initial key. If the key consists of all 0's or all 1's then the shifting of the key at each step accomplishes nothing as the key will remain the same throughout the 16 step process. Another weakness, also associated with the key, is that there exist pairs of keys that will yield identical ciphertext. This means that a different key can be used to decrypt a message from the one that had originally been used. Even with these weaknesses, there are more than 10^{16} different keys that can be used in DES.

In order to demonstrate that DES offered what it believed was an inadequate level of protection, RSA Data Security, Inc. launched its DES Challenge in January 1997. The challenge was to recover a message encrypted using the 56-bit key DES. On June 17 of the same year a university team cracked the message in just 90 days, far less than

what the government had led industry to believe it would take. RSA Data Security responded by launching its DES Challenge II in January 1998. The goal of this second challenge was to see if the 90 day record set during the original challenge could be broken. A team of computer programmers took up the challenge and arranged a coordinated effort among other enthusiasts around the world to crack the encoded message “many hands make light work.” This coordinated effort included 22,000 volunteers linking together over 50,000 CPUs to search for the correct key. In February 1998, after 39 days of searching, the key was found and the message decoded. Still not satisfied, RSA Data Security issued a new challenge which this time was picked up by a team from the Electronic Frontier Foundation (EFF). At this time the government was still claiming that DES was reasonably secure for most applications in industry since it took over a month with thousands of machines to crack it. This would be considered to be beyond what a hostile corporation would reasonably pay to obtain information about a competitor. Some, however, felt that with the appropriate equipment a platform could be developed that would be capable of cracking DES in far less time. The team from EFF set about to build a single customized CPU capable of cracking DES. For less than \$250,000 they built the device and were able in July 1998 to crack the new encrypted message, “It’s time for those 128-, 192-, and 256-bit keys” in only 56 hours. At this point the usefulness of DES for industry applications was drastically questioned since the price of the specialized CPU and the time it took were now in the range of what some corporations might pay to obtain a competitive edge. Proposed solutions to the problem included triple-DES in which the plaintext is in effect encrypted three times or, as the last Challenge message indicated, expanding the key from 56 to 128 or more bits.

14.3.2 IDEA

DES may be reaching a point where it will soon be at the end of its useful life. As a result, there have been several other efforts to develop a new encryption standard. One of these, first developed in 1990 and called the Proposed Encryption Standard (PES), is the International Data Encryption Algorithm (IDEA) [14.10].

IDEA, like DES, is a block cipher operating on 64-bit blocks of plaintext which uses the same algorithm for both encryption and decryption. Unlike DES, however, IDEA uses a 128-bit key. The algorithm itself splits the plaintext into four 16-bit sub-blocks. Each of these blocks then goes through a series of operations which include exclusive-or’s, modulo 2^{16} addition, and modulo $2^{16}+1$ multiplication. The blocks do not, however, go through any complicated permutations. Another difference between IDEA and DES is that IDEA only goes through eight iterations as opposed to DES’s sixteen iterations. The key is split into eight 16-bit sub-blocks (K_1 through K_8). Six of these key sub-blocks are used each iteration, any that are left over are then used in

the next block. After all key sub-blocks are used, the key is shifted 25 bits to the left, and then split up into eight new sub-blocks. After the initial decomposition of the plaintext into four separate blocks (B_1 through B_4), the operations depicted in [Figure 14.3](#) take place during each of the eight iterations [14.10].

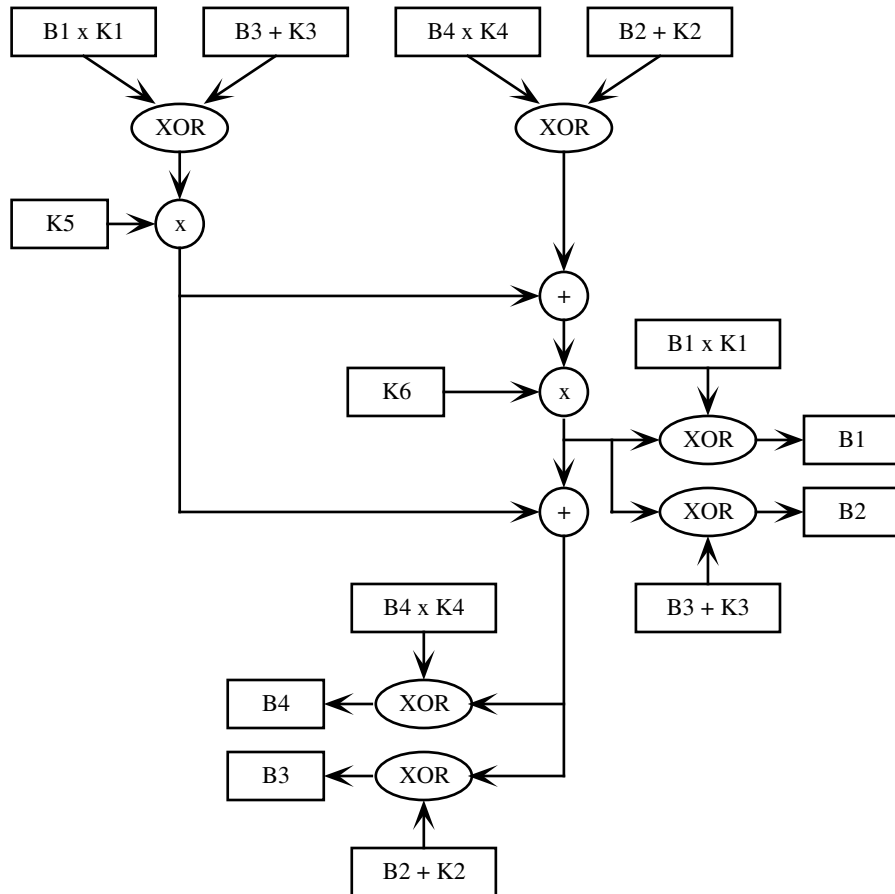


Figure 14.3. IDEA iterative operations.

The final four steps of each iteration (steps 11 through 14) produce the four sub-blocks B_1 through B_4 of the next iteration. After the last iteration the first four operations are performed again and then the four sub-blocks are joined together to form the 64-bit ciphertext. Decryption is done using the same operations except the key

sub-blocks are changed to be either the additive or multiplicative inverses of the original encryption key sub-blocks and are then taken in reverse order.

Analysis of IDEA have shown it to be as fast as DES but, because of its larger key size, a brute force attack will take considerably longer than DES (perhaps greater than 10^{13} years) [14.10]. Another analysis showed that IDEA is also subject to weak key problems. In this case, however, the weak key is one which, if used, can be identified given a *chosen-plaintext* attack (an attack in which the cryptanalyst chooses the text to be encrypted).

14.3.3 Key Escrow and Key Recovery

The White House announced the Escrowed Encryption Initiative in April of 1993. Despite a storm of criticism, NIST announced the Escrowed Encryption Standard less than a year later. The standard, and initiative, were designed to improve the security and privacy of electronic communication while still meeting the needs of law enforcement agencies. At the heart of the debate over Key Escrow are two conflicting desires. The first is the desire of private citizens to protect their electronic communications from eavesdropping. The second is the desire by law enforcement agencies to be able to, after having obtained the appropriate legal permissions, listen in on conversations of those suspected of involvement in illegal activities. Without debating the appropriateness of the initiative, it is informative to understand what is involved in the key escrow process.

Key escrow involves the use of a special encryption chip. Each chip has designed into it a special ‘backdoor’ which will allow a third party to decrypt the transmission. The procedure for utilizing this backdoor entails the use of two unique keys. With only one of the keys the backdoor can’t be opened; it requires both keys. The plan then is to store the two keys at separate “escrow” locations. A law enforcement agency would then have to obtain the appropriate legal permission to perform a wiretap before the keys would be released.

The original chip proposed to implement the key escrow plan was called the *Clipper Chip*. Each chip employs the NSA developed Skipjack algorithm and includes an 80-bit family key F (common to all chips), a serial number S (originally 30 bits), and an 80-bit secret key U which can be used to unlock all messages encrypted by this specific chip [14.1]. Two individuals wishing to use the chip for encryption must also select an 80-bit session key K . The transmitted message consists of two values:

- 1) the encrypted message using the calculation:

$$ciphertext = \text{Encrypt}[K](message) \quad (14.3)$$

- 2) the law enforcement field using the calculation:

$$LE_field = Encrypt[F](Encrypt[U](K) + S) \quad (14.4)$$

The receiver can decrypt the message by simply performing

$$message = Decrypt[K](ciphertext) \quad (14.5)$$

The law enforcement field consists of two parts; an encrypted version of the session key (using the secret key) and the serial number for the chip. A law enforcement agency which has obtained the appropriate legal permissions to perform a wiretap can decrypt the original message in a five step process.

- 1) The law enforcement field is decrypted using the family key to obtain the encrypted session key and the serial number. The calculation is:

$$Encrypted_K_Key + S = Decrypt[F](LE_field) \quad (14.6)$$

- 2) The law enforcement agency provides the serial number for this chip, along with proof of its legal permissions to decrypt the transmission, to the two escrow agents.
- 3) The two escrow agents provide *U1* and *U2*, their parts of the secret key for this chip. These two parts are then XORed to obtain *U*, the unlocking secret key.
- 4) The law enforcement agency can then determine the session key by performing:

$$K = Decrypt[U](Encrypted_K_Key) \quad (14.7)$$

- 5) Now knowing the session key used, the law enforcement agency can decrypt the message by simply performing the same calculation as the receiver:

$$message = Decrypt[K](ciphertext) \quad (14.8)$$

One of the most controversial issues surrounding the key escrow scheme involved the key escrow agents. Obviously the organizations that hold the keys that are used to calculate the unlocking secret key must be trustworthy and reliable. The original intent was to have at least one of these organizations not be part of the federal government. For some, however, there is no organization that they would trust with this tremendous responsibility. The intense debate over the Clipper Chip led to the U.S. government backing down from its initial stand and the Clipper Chip was not adopted as the government had intended.

While the government had backed off some from its Clipper Chip initiative, it had not given up on the concept of key escrow. The Federal Bureau of Investigation, in particular, was concerned with its ability to read the communication of suspects it was trying to bring to justice. In 1997 the director of the FBI told the Senate Committee on Technology that “The problem of encryption is that if we are able to access through a court order the communications of criminals, spies, and terrorists, but we can’t understand [what’s going on], then we are out of business with respect to technology” [14.12]. The new idea being pushed is key recovery. The concept of key recovery was initially proposed by companies interested in protecting encrypted data from the problem with lost keys. The government became interested in the concept because it would allow them to also be able to obtain keys enabling them to read the encrypted communication they had obtained through court ordered wiretaps. An interesting question to ask is whether elements of organized crime or terrorists are actually using encryption to avoid detection by organizations such as the FBI. In a report published in 1997, Dorothy Denning and William Baugh found that the total number of criminal or terrorist cases in which encryption was used to hide information or communication was at least 500 with an estimated annual growth rate between 50 and 100% [14.13].

The debate over whether strong encryption should be available to industry and the public is more than just an issue of privacy versus the government’s desire to crack down on crime. U.S. companies, for example, are concerned that if the US mandates a key recovery system then other nations would also be encouraged to do so and could then listen in on sensitive communication between U.S. companies and their foreign offices. State-sponsored industrial espionage is a fact most industries are aware of and must protect against. Patrick Watson, director of worldwide security for Eastman Kodak, testified before a Senate panel that it “is also important that these concerns over terrorist and criminal activities not be used by foreign governments as a pretext to monitor the legitimate activities of U.S. corporations” [14.12]. He acknowledged that overseas is “where security threats are of greater concern” and testified that export restrictions on technology and encryption techniques had kept his company from being able to use the most capable secure telephone systems. Despite these objections from

industry and a flat rejection by a European commission of a U.S. proposal to establish a global system in which encryption keys would be deposited with an independent organization, the administration continues to push for means to be able to recover encrypted information.

14.3.4 Public Key Cryptography

Public Key or asymmetric cryptosystems differ from symmetric cryptosystems in that they use different keys for decryption and encryption. The concept of public key cryptography was introduced by Whitfield Diffie and Martin Hellman in 1976 [14.2]. The public key process can be expressed as follows:

$$Ciphertext = Encrypt[key1](Plaintext) \quad (14.9)$$

$$Plaintext = Decrypt[key2](Ciphertext) \quad (14.10)$$

There are two modes of public-key system operation. The public key can be used to encrypt the Plaintext, or it can be used to decrypt the Ciphertext. A system which works in only one or the other of these modes is known as an *irreversible public-key cryptosystem*. One that works in both directions is known as a *reversible public-key cryptosystem*. The reason one might use a public-key for decryption is if data origin authentication is desired.

14.3.4.1 Diffie-Hellman Algorithm

The original algorithm presented in “New Directions in Cryptography” [14.2] described a method two individuals could use to generate and distribute a secret key in a public environment. The security of the technique comes from the difficulty of calculating logarithms mod q as compared to multiplication’s mod q (where q is a prime number). In their technique, Diffie and Hellman describe how each user must generate an independent random number X_i from the set of integers $\{1, 2, \dots, q-1\}$ [14.2]. Their explanation is as follows: Each user keeps this chosen value, X_i secret but stores:

$$Y_i = a^{X_i} \bmod(q) \quad (14.11)$$

where a is a large integer, and $a < q$ in a public location. When two users, i and j , wish to communicate securely, they generate a key for their own private use as follows:

$$K_{ij} = a^{x_i x_j} \bmod(q) \quad (14.12)$$

but since neither knows the other's private key X, they must calculate it. User i can calculate K_{ij} by using the public key Y_j and calculating:

$$K_{ij} = (Y_j)^{x_i} \bmod(q) \quad (14.13)$$

$$K_{ij} = (a^{x_j})^{x_i} \bmod(q) \quad (14.14)$$

$$K_{ij} = a^{x_i x_j} \bmod(q) \quad (14.12)$$

User j can obtain the key in a similar manner. For another user to determine the key, however, the following, much more labor intensive, calculation would be required:

$$K_{ij} = Y_i^{((\text{Log}(a))Y_j)} \bmod(q) \quad (14.15)$$

Diffie and Hellman described how a cryptanalyst would require $2^{b/2}$ operations to determine the key if q were chosen so that it was slightly less than 2^b . For a b value of 200 this would result in approximately 10^{30} operations. The Diffie-Hellman algorithm is patented in the United States and Canada until April of 1997.

14.3.4.2 Knapsack Algorithms

A rather interesting algorithm was proposed by Ralph Merkle and Martin Hellman which utilized a famous NP-complete problem for its security [14.3]. The Knapsack problem, simply stated, is: given a collection of objects with different weights, determine which of the items should be placed in the knapsack so that it will weigh a specified amount. The time required to solve this problem grows exponentially with the number of objects in the collection to choose from. The way that Merkle and Hellman utilized this problem for cryptography was to encrypt a message as a solution to a series of knapsack problems. For example, if the objects had weights of 1, 3, 4, 8, 11, and 15, then the cleartext would be grouped into 6-bit blocks and the weights for each of these calculated as follows [14.10]:

Cleartext:	1 1 0 1 0 0	1 0 0 0 1 1	0 0 1 0 1 1
Weights:	1 3 4 8 11 15	1 3 4 8 11 15	1 3 4 8 11 15
Calculation:	1+3+0+8+ 0+ 0	1+0+0+0+11+15	0+0+4+0+11+15
Ciphertext:	=12	=27	=30

The public/private key aspect of this approach lies in the fact that there are actually two different knapsack problems--referred to as the easy and hard knapsack. The easy knapsack approach has been proven to be solvable in $O(n)$ time. It is created as described above with one exception. The sequence of weights must be a **superincreasing sequence**. A superincreasing sequence is one in which each element is greater than the sum of all previous elements in the sequence. An example would be {1, 2, 4, 8, 16, 32}. A hard knapsack sequence is not superincreasing. The hard knapsack sequence is not solvable in $O(n)$ but rather is the NP-complete problem described previously. It is this sequence that is put into the public sector as the public key. The private key will be the easy knapsack sequence and is kept secret. The public key can be generated from the private key by multiplying each element of the easy knapsack sequence by $a \bmod b$, where b is an integer greater than the sum of all numbers in the sequence, and a is an integer which has no factors in common with any value in the sequence. If the easy knapsack sequence were {2, 4, 8, 16, 32} and if a is chosen to be 23 and b to be 65, then the hard knapsack sequence (the public key) would be calculated as follows:

$$\begin{aligned} 2 \times 23 \bmod 65 &= 46 \\ 4 \times 23 \bmod 65 &= 27 \\ 8 \times 23 \bmod 65 &= 54 \\ 16 \times 23 \bmod 65 &= 43 \\ 32 \times 23 \bmod 65 &= 21 \end{aligned}$$

The resulting hard knapsack sequence would be {46, 27, 54, 43, 21}. Using this public key sequence we can encrypt the following 10-bit message as follows:

Cleartext:	1	0	0	1	1	0	0	1	0	1
Public key:	{46	27	54	43	21}	{46	27	54	43	21}
Calculation:	46+	0+	0+	43+	21=110	0+	0+	54+	21+	0=75
Ciphertext:	110,75									

The message can then be decrypted using the private key. The first step of this decryption is to determine a^{-1} so that $a \times a^{-1} = 1 \bmod b$. With $a = 23$ and $b = 65$, $a^{-1} = 17$. Each value in the ciphertext is then multiplied by this number mod b which will yield the plaintext values.

Ciphertext:	110,75
Private Key:	{2, 4, 8, 16, 32}
	$110 \times 17 \bmod 65 = 50 = 10011 \text{ (} 2+0+0+16+32 \text{)}$
	$75 \times 17 \bmod 65 = 40 = 00101 \text{ (} 0+0+8+ 0+32 \text{)}$
Plaintext:	10011 00101

In practice, the value of b is chosen to be very long as is each value in the knapsack (approximately 200 to 400 bits). The knapsacks themselves should contain about 200 items [14.10]. Even with values of this length, this knapsack algorithm has been broken (a method was shown that allowed the private key to be determined from the public key). Many other proposed knapsack algorithms have arisen but the wisest course of action is to probably stay away from using them.

14.3.4.3 RSA

Named after its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, RSA has proven to be an extremely reliable algorithm used for both public key encryption and digital signatures (discussed in the next section). The security of RSA is derived from the problem associated with the factoring of large numbers [14.8]. The keys used are calculated from a pair of large prime numbers p and q . These prime numbers may exceed 200 digits in length. The public key consists of a pair of numbers, n and e , where $n = p \times q$. The value for e , which is the encryption key, is chosen randomly and should be relatively prime to $(p - 1) \times (q - 1)$. The decryption key, d , is then computed as $d = e^{-1}(\text{mod } (p - 1) \times (q - 1))$ [14.10]. The two prime numbers chosen earlier, p and q , are no longer needed and can be discarded but should be kept secret.

To encrypt a message m using the RSA algorithm, the message should be partitioned into blocks of characters (or bits for binary data) no larger than the number of characters or bits in n . Each of these blocks will be encrypted into ciphertext blocks, c_i (of the same size), using the encryption key (public key) and the following formula [14.8, 14.10]:

$$c_i = \text{Encrypt}(m_i) = m_i^e (\text{mod}(n)) \quad (14.16)$$

Decryption uses the decryption (private) key as follows:

$$m_i = \text{Decrypt}(c_i) = c_i^d (\text{mod}(n)) \quad (14.17)$$

In actuality the values for d and e are interchangeable so that encryption could take place with d and decryption with e . An example will help to illustrate the RSA encryption and decryption processes. If $p = 31$ and $q = 53$, then $n = 1643$. The encryption key e must be relatively prime; that is, have no factors in common with $(p - 1) \times (q - 1) = 30 \times 52 = 1560$. We can randomly select e to be 71. The public key is thus the pair $\{71, 1643\}$. The decryption key is:

$$d = 71^{-1} (\text{mod } 1560) = 791$$

Remember that the inverse function here is a modulo inverse which means that:

$$71 \times 791 = 1 \pmod{1560}$$

To encrypt:

$$m = 34511228919$$

the message must first be broken into blocks of less than four characters since n has four characters (we must insure that there is a unique value for each block mod n). The corresponding m_i blocks and their encrypted forms would then be

$$\begin{array}{ll} m_1 = 345 & c_1 = 345^{71} \pmod{1643} = 190 \\ m_2 = 112 & c_2 = 112^{71} \pmod{1643} = 196 \\ m_3 = 289 & c_3 = 289^{71} \pmod{1643} = 81 \\ m_4 = 19 & c_4 = 19^{71} \pmod{1643} = 475 \end{array}$$

The ciphertext is {190 196 81 475}. The process to decrypt the ciphertext as explained before would produce:

$$\begin{array}{l} m_1 = 190^{791} \pmod{1643} = 345 \\ m_2 = 196^{791} \pmod{1643} = 112 \\ m_3 = 81^{791} \pmod{1643} = 289 \\ m_4 = 475^{791} \pmod{1643} = 19 \end{array}$$

RSA works because there is no current way known to solve the factorization problem in polynomial time. If there were, then RSA would not be secure. RSA has been tested and considerably researched with no serious flaws found. This does not guarantee its security but rather provides a certain level of trust in it. One known problem with RSA is illustrated below. It has been shown that for every key combination there exists a message such that:

$$m^e = m \pmod{n} \quad (14.18)$$

which would mean the encrypted message and the original message are equivalent [14.9]. For example, using the values for e and n obtained before, if the message was {159}, the ciphertext would be:

$$c = 159^{71}(\bmod 1643) = 159$$

The encryption process returned the same value as the original message as does the decryption process as is illustrated below:

$$m = 159^{791}(\bmod 1643) = 159$$

Another problem with RSA is that it is considerably slower than DES for large quantities of data. Despite these problems, RSA is a *de facto* standard for much of the world. The International Organization of Standards has even selected RSA as the standard to be used for digital signatures.

14.3.5 Digital Signatures

A digital signature is completely analogous to a handwritten signature used for centuries to authenticate documents. The reason that signatures have been used is that, generally speaking, everyone's signature is unique and hard to forge and a document which has been signed would be hard to later repudiate. These same assurances are desirable for computer generated and transmitted documents. We would like to be able to prove that a document sent to us by somebody was indeed sent to us by that individual (authentication) and we would also like to later be able to prove that it had been sent to us by that individual (non-repudiation). A digital signature is designed to provide these assurances.

One method to implement digital signatures is to use encryption with pairs of known and secret keys and a trusted arbitrator. Two individuals who want to communicate each share a different secret key with the arbitrator. The sender encrypts the message and sends it to the arbitrator who then decrypts it using the key known only to it and the sender. The arbitrator then encrypts the cleartext message with the secret key known only to it and the receiver and sends the new ciphertext on. The receiver can then decrypt the message using its key. Obviously the arbitrator plays a key role and all parties must implicitly trust this third party. There are a number of drawbacks to this approach. First, there is the obvious need for a trusted third party. In addition there is a considerable amount of overhead since each message must be encrypted, decrypted, and transmitted twice. An alternative approach is to instead use a public key system, such as RSA, with senders using their private key to encrypt the message and receivers using the public key to decrypt. Obviously, since only the senders would know their private key, this assures the receiver that the sender did in fact author the message. Often, since encryption takes a considerable amount of time, a sender may not want to encrypt the entire document but rather provide just a signature

authentication. Often digital signature algorithms simply encrypt the signature with an additional time stamp and hash value for authentication purposes.

14.3.5.1 The Digital Signature Standard (DSS)

In 1991, NIST proposed that their Digital Signature Algorithm (DSA) be accepted as the Digital Signature Standard (DSS) for the United States [14.6]. This proposal was immediately met with opposition, mostly from proponents of RSA. DSA is a variation of two earlier signature algorithms by Schnorr and ElGamal [14.10]. In DSA the public key is a set of four values $\{p, q, r, t\}$ and the private key is a single value x . (for a full description of DSA see [14.6, 14.10]). If the sender wants to provide a signature for a message m , the entire message is not used as part of the signature. Instead, the Secure Hash Algorithm [14.5] is used to create a 160-bit output referred to as the “message digest”. If the message is altered, the message digest of this new copy will not match the message digest of the original. Thus, attaching the original message digest provides a comparable level of assurance as attaching the entire message but without the extra overhead. Obviously, since we are talking about a hashing function, two different messages may hash to the same message digest. The 160-bit size of the output, however, makes it computationally infeasible to find two messages with the same message digest. As was mentioned, DSA has received much criticism, most of which comes from the RSA community. One valid criticism of DSA, however, is that it is slower than RSA. Otherwise, it provides a reasonable level of security for authentication.

14.3.5.2 ESIGN

Another digital signature algorithm is ESIGN. This algorithm is reported to be both faster than RSA and DSA with comparable key and signature lengths, and at an equivalent level of security [14.7, 14.10]. ESIGN uses a pair of large prime numbers, p and q , for its secret key and a public key, n , which is computed to be p^2q . Like DSA, ESIGN uses a hash function, $H(m)$, on the message, m , to provide an equivalent block to the message digest. To compute the signature, the sender first selects a value x such that $x < pq$, and determines w which is the smallest integer that is larger than $(H(m) - x^k \bmod n)/pq$ [14.10]. The value for k , as recommended by the developers of the algorithm, should be a power of two between 8 and 1024. The signature, s , is then calculated as follows [14.10]:

$$s = x + \left((w / kx^{k-1}) \bmod p \right) pq \quad (14.19)$$

The receiver can verify this signature by first computing $s^k \bmod n$ and a which is the smallest integer that is larger than the number of bits in n divided by 3. The signature is considered authentic if both of the following conditions hold true [14.10]:

$$H(m) \leq s^k \bmod n \quad (14.20)$$

$$s^k \bmod n < H(m) + 2^a \quad (14.21)$$

When ESIGN was first developed, the value for $k = 2$ and $k = 3$ were quickly broken. When the current recommended values (a power of two greater than 4) have been used, the algorithm has not been broken. In addition, the size of p and q are recommended to be at least 192 bits long which will result in a value of n with at least 576 bits (192×3). This seems to provide adequate security for the algorithm.

14.4 PGP—Pretty Good Privacy

PGP, which stands for “Pretty Good Privacy,” is a program which uses encryption to help individuals maintain the privacy of their email and files. It is also designed to provide a digital signature system to allow users to electronically sign documents and verify other people’s electronic signatures. It is a public key system which is widely available for downloading from the Internet. PGP was developed by Phil Zimmerman in 1991. He gave it to a friend who made it available for free on the Internet. The problem with releasing it in this manner was that Zimmerman didn’t have a license for either the RSA or Merkle-Hellman patents which it employed. Many on-line service providers urged their customers to not use PGP because of the legal problems surrounding the patents, but the popularity of it nonetheless increased. Interest in PGP increased overseas at a tremendous rate and soon versions of it were available in Europe, Japan, and Australia—this despite the export laws that prohibited the transfer of cryptographic software outside of the U.S. In 1994, a version of PGP based on the RSAREF toolkit was developed and released. The RSAREF toolkit contained a license which allowed use of the algorithms in software provided the resulting software was for noncommercial use only. At last an official version of PGP could be released. Besides the controversy over the use of RSA in PGP, Zimmerman also faced federal charges for exportation of cryptographic programs for several years.

Originally charged in 1993, Zimmerman was finally cleared when the government, with very little explanation, dropped all charges against him in 1996.

PGP uses a command line interface to communicate with it. To run it you need to only type *pgp*. Typing *pgp* and a filename will allow you to decrypt a file. Using the *-c* flag allows you to encrypt the file. Using the *-kv* flag allows you to view all of the public keys in a key ring. A PGP key ring is a file which contains the public keys for people that you frequently want to communicate with. Some of the more common commands used in PGP are:

<code>pgp <file></code>	decrypts <file>
<code>pgp -c <file></code>	encrypts <file>
<code>pgp -h</code>	displays help information
<code>pgp -kg</code>	generate a PGP key
<code>pgp -kv</code>	view keys
<code>pgp -kr</code>	remove key
<code>pgp -e <file> <userid></code>	encrypt <file> with <userid>
<code>pgp -ea <file> <userid></code>	encrypt <file> with <userid> into ASCII
<code>pgp -s <file></code>	digitally sign document <file>

14.5 Public Key Infrastructure

It is not uncommon in today's heavily networked environment for individuals to have several passwords and userids. Passwords may be needed to gain initial access to a system and again for certain applications and services. In certain situations these passwords may even be sent in an unencrypted manner over the network. At the same time we are also seeing an increasing use of the network for commerce. In order to safely transact business across the network, however, a system is needed to encrypt transactions to keep them confidential and also to verify the identity of each individual involved in the transaction. As was seen earlier, public key cryptography can be used to perform both functions but it relies on each side being able to obtain the public key of the other party. How does each party know that it has truly received the public key for the other individual and not the key from somebody claiming to be the other party? What is lacking is a trusted entity who both can rely on to verify the other's authenticity. This is the purpose of a Public Key Infrastructure.

At the heart of a Public Key Infrastructure (PKI) is the digital certificate. A digital certificate is used to verify the identity of users, servers, and other objects on a network. The X.509 standard has been created for digital certificates. While the standard itself has existed for sometime, the infrastructure used to support digital certificates has been much slower to evolve. In order to understand the need for this infrastructure, an analogy sometimes used is to compare the digital certificate with a

driver's license. A license is used to uniquely identify an individual. We have faith in the license because we trust the organization that issued it. In addition, the license can be obtained by anybody in many locations, all of which trust each other and honor the licenses issued by the others. Digital certificates also uniquely identify a user but are only valuable if we can trust the issuing agency as we do in the case of a driver's license. We should be able to query the issuing organizations to verify the authenticity of a certificate and to also obtain a certificate ourselves when needed. An infrastructure to accomplish these purposes is what is needed in the public key environment.

There are a number of advantages to be gained from the creation of a wide-spread PKI. The use of digital signatures provides a means to create a secure link between a client and a server for a business. For applications such as banking-at-home, online brokerage services, and any transaction using a credit card for payment, a secure link is essential. For large corporations in which users may require access to a number of different servers but do not want to memorize multiple passwords and userids, PKI provides a method to allow a single login while still ensuring authentication across the network. Overall, PKI can become the backbone of a secure corporate enterprise. As electronic commerce continues to proliferate, the use of PKIs will become more widespread.

14.6 Steganography

An area similar to cryptography in its purpose but slightly different in its application is Steganography. Steganography is the art of obscuring information in a manner so as to prevent its detection. While the information that is being hidden can be in any form, steganography has gained notoriety because of its use by individuals attempting to conceal pornographic images inside of other legitimate images. To the casual observer the file appears to only contain the legitimate image and it requires special software (freely available on the Internet) to extract the hidden image. The image has not been altered in the sense that encryption alters a file, rather it is simply hidden from view. To understand how steganography works requires an understanding of how images are stored on a computer.

Common methods to store computerized images involve the encoding of individual picture elements (pixels). A pixel is nothing more than one small point of an image. Each image is made up of many thousands or millions of these pixels. Standard image sizes are 640 X 480 pixels, 800 X 600 pixels, and 1024 X 768 pixels. Depending on the number of colors and shades are possible for each pixel, the amount of information stored for each pixel may range from 8 to 24 bits. With 8 bits, for example, each pixel can take on 1 of 256 different colors ($2^8 = 256$). If 24 bits are used over 16 million colors can be represented. To illustrate how images can be hidden

inside of other images, imagine a simple black-and-white image. If we use 1 bit to represent each pixel we will save a tremendous amount of space but the image will have only two colors, black or white. A 0, for example, could be used to indicate a white pixel and a 1 could represent a black pixel. An image using this method would appear very jagged as there would be no smooth transition as we change from one color to the other. If we instead used 2 bits to represent each pixel we can now have 4 colors (or shades) as follows:

```
00 White
01 Light Grey
10 Dark Grey
11 Black
```

This new scheme will tremendously improve our ability to ‘soften’ our image but still has a long ways to go to eliminate the ‘jagged’ edges that would result. If we again increased the number of bits used to represent each pixel we could increase the number of shades and further soften the image. With 8 bits, there could be 254 different shades of gray between pure white and pure black. The difference between two consecutive shades will be very slight and very hard to detect by most individuals. This is the key to steganography using images. If we use the least significant bit to transport our hidden message or image, the change that would be made in the original would be very slight and could not be detected by most individuals. For example, imagine an image which included the following 8 bytes of information representing 8 consecutive pixels:

```
00101001
00101001
00101010
00101100
00101001
00111110
01010101
01110010
```

If we wanted to hide a message in this image which started with the letter A (01000001 in ASCII), we would simply take each bit of the letter in order and place them in the least significant bit location for each consecutive byte representing the different pixels. Our example above with the hidden letter A would then be:

```
00101000
00101001
00101010
00101100
00101000
00111110
01010100
01110011
```

There is a difference between the two sequences but the change to the resulting image would be so slight as to be unnoticeable. Obviously it will require 8 bytes of picture image to hide a single character using this scheme. A 640 X 480 pixel image using 8 bits for color, however, will require 307,200 bytes of information and could thus hide a file containing 38,400 characters. It could also hide a 300 X 240 pixel image using 4 bits for color using the same technique. If instead of an 8-bit color scheme the original image utilized a 24-bit color scheme, each pixel would require 3 bytes of information and the resulting image would contain 921,600 bytes. With a 24-bit color scheme the difference between the various shades is so small that 3 bits of each 24-bit pixel encoding could be used to hide information without a drastic degradation in the original image. This would mean that we could actually hide a file of 345,600 characters or a 640 X 480, 8-bit color image inside of the original image. Care must be taken in choosing the original image (sometimes referred to as the *container*) as an image with large areas of solid color would be noticeably changed if another image or message were hidden inside of it.

Steganography is not limited to modification of images on computers in order to hide information. Text, sound, and binary files on different media could be used as well. Steganography, for example, could be used to hide information inside of an ordinary cassette recording without causing a change that could be discernable to human hearing. Steganography by itself does not ensure secrecy—especially if common steganographic tools are used. If, however, the message or image to be hidden is itself encrypted before being placed inside of a container, the resulting file becomes much more secure.

While, as indicated before, the use of steganography has received some notoriety because of its use by pornographers to send illicit materials, this is by no means the only use of it. It is a common technique today to include a *digital watermark* in images used online so as to be able to track copyrights. It is a simple matter to include an individual's name and the date in the first few dozen bytes of an image before it is displayed on the Internet so that it can later be extracted if an argument over original ownership of the image occurs.

14.7 Summary

Cryptography is as valuable in today's highly technical society to help secure our communication as it was for the past several thousand years. It entails the disguising of messages so that only the originator and the intended recipient(s) will be able to understand the contents. Two basic techniques used in cryptography are substitutions and transpositions. In substitutions, each character is replaced by another so that the message is obscured. In transpositions, the characters remain the same but their order is changed to accomplish the obscuring. Encryption schemes usually employ some sort of key, known only to the sender and receiver, to describe what substitutions and transpositions to perform.

Modern day algorithms used to encrypt our digital communications use a combination of these techniques. The Data Encryption Standard (DES), for example, uses logical operations such as XOR's to change the characters (bits) and at the same time divides the blocks up, shuffling them around and thus transposing their positions. DES suffers from a number of criticisms and many individuals are suspicious of its origins and ability to provide security. Other algorithms, such as IDEA, have been developed which use larger keys to increase the complexity of breaking the scheme.

In public-key algorithms, each individual uses two separate keys. The public key is published openly and is used by those wishing to send a message to the individual whose public key was used. The individual whose public key was used has a second key, known as the private key, which is secret and is used to decrypt the message. Variations of public-key systems are also used to provide digital signatures for authentication and non-repudiation purposes.

14.8 Projects

- 14.1 Using [Table 14.1](#), decrypt the following ciphertext which was encrypted using a simple substitution scheme:

BXFMM SFHVM BUFEN JMJUJ BCFJO HOFDF
TTBSZ UPUIF TFDVS JUZPG BGSFF TUBUF
UIFSJ HIUPG UIFQF PQMFU PLFFQ BOECF
BSBSN TTIBM MOPUC FJOGS JOHFE

What effect did grouping the letters have on the decryption?

- 14.2 Given the key SECRET and the following ciphertext:

YSCZV YGVEV FXSX CUQR

determine what method was used to encrypt the message and decrypt it.

- 14.3 Given the following ciphertext:

ASDFGHJKLQWERTYUIOZXCV

Find a key to be used with the Vigenere Table that will yield the following plaintext:

FOUR SCORE AND SEVEN YEARS

Find a key that, given the same ciphertext, will decrypt to:

MARES EAT OATS AND DOES EAT

What are the implications, in terms of security, of ciphertext which is no longer than the key used to encrypt it?

- 14.4 An interesting problem, similar to the famous “Dining Philosophers Problem” in texts on operating systems, is the “Dining Cryptographers Problem” by David Chaum (see “The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability”, *Journal of Cryptology*, Vol. 1, No. 1, 1988, pp. 65-75). The problem begins at a restaurant in which three cryptographers are dining. The waiter shows up and announces that the bill has been paid for, anonymously. The diners talk among themselves and the question is raised as to whether it is one of them that is paying for the meal or somebody else. Develop an algorithm which will allow them to determine whether one of the cryptographers paid the bill or if somebody else did. Should the generous individual be one of the cryptographers, the algorithm should not indicate who it is (i.e., the generous cryptographer must remain anonymous). The answer should not involve writing anything down (they would recognize each other’s handwriting) nor should it involve a fourth party (you just can’t trust somebody else to remain silent).

- 14.5 A common problem in cryptography is the distribution of keys. Obviously one way to distribute them is to have a courier deliver them by hand. This is expensive and time consuming so an electronic means is

much more desirable. How can public-key schemes be used to help solve the key distribution problem?

- 14.6 Given a knapsack problem with the following collection of weights {14, 27, 32, 41, 57, 61, 74, 91, 107, 118}, which items are actually in the knapsack if its total weight is 254?
- 14.7 A number of newspapers and periodicals provide a daily crypto problem for their readers enjoyment. The problem generally employs a simple substitution method. Develop a computer program to help an individual solve one of these simple crypto problems. The program should allow the user to make various substitutions, and should provide suggestions, base on letter frequency, when requested.

14.9 References

- 14.1 Alexander, Michael “Controversy Clouds New Clipper Chip”, *InfoSecurity News*, July/August 1993, pp. 40-42.
- 14.2 Diffie, Whitfield and Hellman, Martin, “New Directions in Cryptography”, *IEEE Transactions on Information Theory*, Vol. 22, No. 6, November 1976, pp. 644-654.
- 14.3 Merkle, R.C. and Hellman, M.E., “Hiding Information and Signatures in Trapdoor Knapsacks”, *IEEE Transactions on Information Theory*, Vol. 24, No. 5, September 1978, pp. 525-530.
- 14.4 National Bureau of Standards, “Data Encryption Standard”, *FIPS PUB 46*, January 15, 1977.
- 14.5 National Institute of Standards and Technology, “Secure Hash Standard”, *FIPS PUB 180*, May 11, 1993.
- 14.6 National Institute of Standards and Technology, “Digital Signature Standard”, *FIPS PUB 186*, May 19, 1994.
- 14.7 Okamoto, T. “A Fast Signature Scheme Based on Congruential Polynomial Operations”, *IEEE Transactions on Information Theory*, Vol. 36, No. 1, 1990, pp. 47-53.

- 14.8 Rivest, R.C.; Shamir, A. and Adelman, L., “A Method for Obtaining Digital Signatures and Public-Key Cryptosystems”, *Communications of the ACM*, Vol. 21, No. 2, February 1978, pp. 120-126.
- 14.9 Sauder, T. and Ho, C.Y., “An Analysis of Modern Cryptosystems”, Technical Report CSC-93-11, Department of Computer Science, University of Missouri - Rolla, 1993.
- 14.10 Schneier, Bruce, Applied Cryptography, John Wiley & Sons, Inc., New York, New York, 1994.
- 14.11 Stallings, William, Network and Internetwork Security Principles and Practice, Prentice Hall, Englewood Cliffs, New Jersey, 1995
- 14.12 Braun, David, “FBI Insists on Crypto Access”, TechWire, available for download from the Internet at www.techweb.com.
- 14.13 Denning, Dorothy, and Baugh, William; “Encryption and Evolving Technologies as Tools of Organized Crime and Terrorism”, US Working Group on Organized Crime, National Strategy Information Center, June 1997.

14.10 Extended Bibliography

- 14.14 Akl, Selim, “Digital Signatures: A Tutorial Survey”, *Computer*, Vol. 16, No.2, February 1983.
- 14.15 Anderson, Ross J., “Why Cryptosystems Fail”, *Communications of the ACM*, Vol. 37, No. 11, November 1994, pp. 32-40.
- 14.16 Bhimani, Anish, “All Eyes on PKI”, *Information Security*, October 1998, pp. 22-31.
- 14.17 Blackburn, S.; Murphy, S. and Stern, J., “Weaknesses of a public-key cryptosystem based on factorizations of finite groups”, *EUROCRYPT 93*, May 1993, pp. 50-54.

- 14.18 Campbell, C.M., "The Design and Specification of Cryptographic Capabilities", *IEEE Computer Society Magazine*, Vol. 16, No. 6, November 1978, pp. 15-19.
- 14.19 Chaum, David, "The Dining Cryptographers Problem: Unconditional Sender and Receiver Untraceability", *Journal of Cryptology*, Vol 1., No. 1, January 1988, pp. 65-75.
- 14.20 Coppersmith, D., "The Data Encryption Standard (DES) and its strength against attacks", *IBM Journal of Research and Development*, Vol. 38, No. 3, May 1994, pp. 243-250.
- 14.21 d'Agapeyeff, Alexander, Codes and Ciphers, Oxford University Press, New York, New York, 1939.
- 14.22 Demillo, Richard and Merritt, Michael, "Protocols for Data Security", *Computer*, Vol. 16, No. 2, February 1983, pp. 39-50.
- 14.23 Denning, Dorothy, Cryptography and Data Security, Addison-Wesley, Reading, Massachusetts, 1982.
- 14.24 Denning, Dorothy, "The Clipper Encryption System", *American Scientist*, Vol. 81, No. 4, July/August 1993, pp. 319-324.
- 14.25 Diffie, W. and Hellman, M.E., "Privacy and Authentication: An Introduction to Cryptography", *Proceedings of the IEEE*, Vol. 67, No. 3, March 1979, pp. 397-427.
- 14.26 Fahn, Paul, "Answers to Frequently Asked Questions about Today's Cryptography", RSA Laboratories, September, 1992, available via anonymous *ftp* at *csrc.nist.gov*, *pub/secpub/faq-k.ps*.
- 14.27 Fisher, Sharon, "Encryption Policy Spurs Concern", *Communications Week*, April 26, 1993, pg. 8.
- 14.28 Fisher, Sharon, "Who'll Hold Clipper Keys?", *Communications Week*, September 27, 1993, pp. 35-36.
- 14.29 Folkers, Richard, "Jimmying the Internet: Why the U.S. encryption standard is very vulnerable", *U.S. News & World Report*, September 14, 1998, pp. 45-46.

- 14.30 Ford, Warwick, Computer Communications Security, Prentice Hall, Englewood Cliffs, New Jersey, 1994.
- 14.31 Garfinkel, Simson, PGP: Pretty Good Privacy, O'Reilly & Associates, Inc., Sebastopol, CA, 1995.
- 14.32 Gasser, Morrie, Building A Secure Computer System, Van Nostrand Reinhold, New York, New York, 1988.
- 14.33 Glover, D. Beard, Secret Ciphers of the 1876 Presidential Election, Aegean Park Press, Laguna Hills, California, 1991.
- 14.34 Gong, Li, "Authentication, Key Distribution, and Secure Broadcast in Computer Networks Using No Encryption or Decryption", SRI International Report SRI-CSL-94-08, May 1994.
- 14.35 Hart, George W., "To Decode Short Cryptograms", *Communications of the ACM*, Vol. 37, No. 9, September 1994.
- 14.36 Hellman, M.E., "The Mathematics of Public-Key Cryptography", *Scientific American*, Vol. 241, No. 8, August 1979, pp. 146-157.
- 14.37 Hoffman, Lance and Faraz, Ali, "Cryptography Policy", *Communications of the ACM*, Vol. 37, No. 9, September 1994.
- 14.38 Johnson, Neil F., and Jajodia, Sushil, "Steganalysis of Images Created Using Current Steganography Software", available at www.isse.gmu.edu/~njohnson/ihws98/jjgm.html, 10/31/1998.
- 14.39 Johnson, Neil F., and Jajodia, Sushil, "Exploring Steganography: Seeing the Unseen", *IEEE Computer*, Vol 31, No. 2, February 1998, pp. 26-34.
- 14.40 Landau, Susan; Kent, Stephen; Brooks, Clint; Charney, Scott; Denning, Dorothy; Diffie, Whitfield; Lauck, Anthony; Miller, Douglas; Neumann, Peter and Sobel, David, "Crypto Policy Perspectives", *Communications of the ACM*, Vol. 37, No. 8, August 1994, pp. 115-121.
- 14.41 Levitt, Jason, "The Keys to Security", *Information Week*, Aug, 31, 1998, pp. 51-60.

- 14.42 McClure, Stuart, "PKI tames network security", *Infoworld*, September 14, 1998, pp. 65-66.
- 14.43 Nechvatal, James, Public-Key Cryptography, NIST Special Publication 800-2, April 1991.
- 14.44 Okamoto, T. and Shiraishi, A., "A Fast Signature Scheme Based on Quadratic Inequalities", *Proceedings of the 1985 Symposium on Security and Privacy*, IEEE, April 1985, pp. 123-132.
- 14.45 Patterson, Wayne, Mathematical Cryptology for Computer Scientists and Mathematicians, Rowman & Littlefield, Pub., Totowa, New Jersey, 1987.
- 14.46 Schneier, Bruce, "The Blowfish Encryption Algorithm", *Dr. Dobb's Journal*, Vol. 19, No. 4, April 1994, pp. 38-40.
- 14.47 Schneier, Bruce, "The Cambridge Algorithms Workshop", *Dr. Dobb's Journal*, Vol. 19, No. 4, April 1994, pp. 18-24.
- 14.48 Schneier, Bruce, "The IDEA Encryption Algorithm", *Dr. Dobb's Journal*, Vol. 18, No. 12, December 1993, pp. 50-56.
- 14.49 Selmer, Ernst, "From the Memoirs of a Norwegian Cryptologist", *EUROCRYPT 93*, Lofthus, Norway, May 1993, pp. 142-150.
- 14.50 Simmons, Gustavus, "Subliminal Communication is Easy Using the DSA", *EUROCRYPT 93*, Lofthus, Norway, May 1993, pp. 218-232.
- 14.51 Simmons, Gustavus J., "Cryptanalysis and Protocol Failures", *Communications of the ACM*, Vol 37, No. 11, November 1994, pp. 56-65.
- 14.52 Smith, Laurence D., Cryptography: The Science of Secret Writing, W.W. Norton & Company, Inc. New York, New York, 1943.
- 14.53 Smith, Peter, "Cryptography Without Exponentiation", *Dr. Dobb's Journal*, Vol. 19, No. 4, April 1994, pp. 26-30.
- 14.54 Stallings, William, "SHA: The Secure Hash Algorithm", *Dr. Dobb's Journal*, Vol. 19, No. 4, April 1994, pp. 32-34.

15

MALICIOUS CODE

It's March 14, 11:55 PM. A group of business partners are putting the finishing touches on an important report. After celebrating the completion of their efforts they identify a previously unnoticed typo. At 12:05 AM they turn the computer back on only to be greeted with a message saying "Beware the Ides of March." The hard drive spins furiously and the report is deleted by a computer virus.

Computer viruses are just one example of what is commonly referred to as *malicious code*, or *malicious programs*. Malicious programs are created to perform a series of harmful actions on a computer system. Examples of some actions include file deletion, file corruption, data theft, and the less harmful but equally annoying practical joke. These programs often remain dormant and hidden until an activation event occurs. Examples of activation events are program execution and specific access dates, such as March 15, system reboot, and file access. When the predetermined activation event occurs, the malicious program begins its task. In the example above, this task was the deletion of all files in the computer system.

The type of malicious behavior that may be exhibited is not limited to computer viruses. Other types of malicious programs are worms and Trojan horses. The difference between each of these programs is in how they are introduced to, or *infect*, a computer system. This chapter discusses viruses, worms and Trojan horses; and further explores their differences and similarities.

15.1 Viruses

One of the better known types of malicious code is the virus. Named after its biological counterpart, a virus is a segment of code that attaches itself to existing programs and performs some predetermined action when the host program is executed. These actions typically include system or file modification and further infection. The

additional modification often takes the form of a malicious action such as file alteration or disk erasure. For the action to take place, the virus code must be activated. As viruses are not stand alone programs, they require a host application or operating system for activation. Viruses only infect executable programs and not data as a virus requires a means of activation which a data file cannot provide.

15.1.1 Infection

Before a system can be infected by a virus, the virus must be created, and then introduced to the system. It used to be that an in-depth knowledge of an operating system was needed to create a virus [15.3]. Depending on the operating system, a knowledge of program execution techniques, file system management and storage, and operating system-program interaction and communications was necessary [15.8, 15.9]. This type of knowledge, however, is no longer required as numerous virus creation programs are widely available on the Internet.

For a computer system to become infected with a virus, the system must come into contact with a carrier of the virus. Contact can come from many sources. For example, a co-worker can bring an infected disk from home, or a student can bring home a disk that was infected at a school lab. Obviously, any communal computing environment is prone to infection. The potentially large number of users, many of whom are careless, promotes the infection and spread of a computer virus. For these environments to become infected and a virus to have significant impact, however, the virus must be well placed in the environment.

For a virus to have a wide spread effect on the computing community, the virus must be strategically placed. It must be placed in a location that is guaranteed to be accessed by numerous people—preferably a location that will promote large scale, if not world wide, distribution. Some very popular locations to place applications that are infected with the virus are public and private computer bulletin boards, on-line computer services, and file repositories (i.e., public *FTP* sites) [15.3]. On occasion, authors have been able to place their viruses in commercially available programs before they were shipped for sale to the public [15.7]. An otherwise isolated system would then become infected without the computer operator either knowing or considering the system susceptible to infection. Viruses are not limited to placement on officially licensed software. Not surprisingly, viruses are often found on software that has been pirated—illegally copied without permission by the publisher [15.7]. Because many pirated copies of software packages, particularly games and the more popular operating system, are easily found and widely distributed, they make excellent programs to infect [15.3].

After a system, or program, is infected, it becomes a carrier that can infect other systems. This is accomplished by executing an infected program on the uninfected

system. Thus, a user can take an infected program from one location and infect another system by executing the program at a second location. In situations where the media carries a copy of the operating system, simply accessing the media may result in infection. One such system is the Disk Operating System employed by the Apple 2 series of computers. These particular systems initialized a disk with operating system information that was executed whenever the disk was accessed. Thus, simply requesting a listing of a disk's contents would result in the execution of a disk stored operating system program, and infection of the system. It was therefore possible to quickly and easily install viruses that would do something as simple as initialize a disk after a specific number of accesses.

15.1.2 Theory behind Viruses

To better understand what a virus can do, it is helpful to understand how a virus performs its task. [Figure 15.1](#) contains a flow chart describing the basic actions of a virus. Each action identified in the figure is numerically labeled for explanatory purpose only. The order in which these actions are performed is indicated both pictorially, in [Figure 15.1](#), and in the discussion which follows. While the order of these actions may vary with each virus, the general process remains unchanged—perform an undesirable task and infect other programs and operating systems.

For this discussion two terms are defined: virus activation and virus execution. Virus activation will refer to the initiation of the virus. Virus execution, however, will refer to the initiation of the portion of the virus that performs the possibly harmful activity the code not directly concerned with infecting a system. Thus, virus execution must be preceded by activation, but activation may not necessarily lead to execution.

Once an infected program is executed or an infected operating system performs a task, the virus is activated. The virus will first determine whether it should be executed. In the event that it should not be executed, it will attempt to locate and identify other susceptible programs, disks, or systems. Any such item will then be infected. After determining that all susceptible items carry a copy of the virus, the virus will stop and allow normal operations to proceed. If the virus meets all of its internal conditions to execute, it will do so. Upon completion of execution, the virus may either reset its conditions or remove itself. The virus will complete and allow normal computation to continue. While this process may seem time consuming and obvious to the user, it is not. Computers operate so quickly nowadays that this process may go easily unnoticed, and often does. With the assistance of [Figure 15.1](#), the virus life cycle is depicted below.

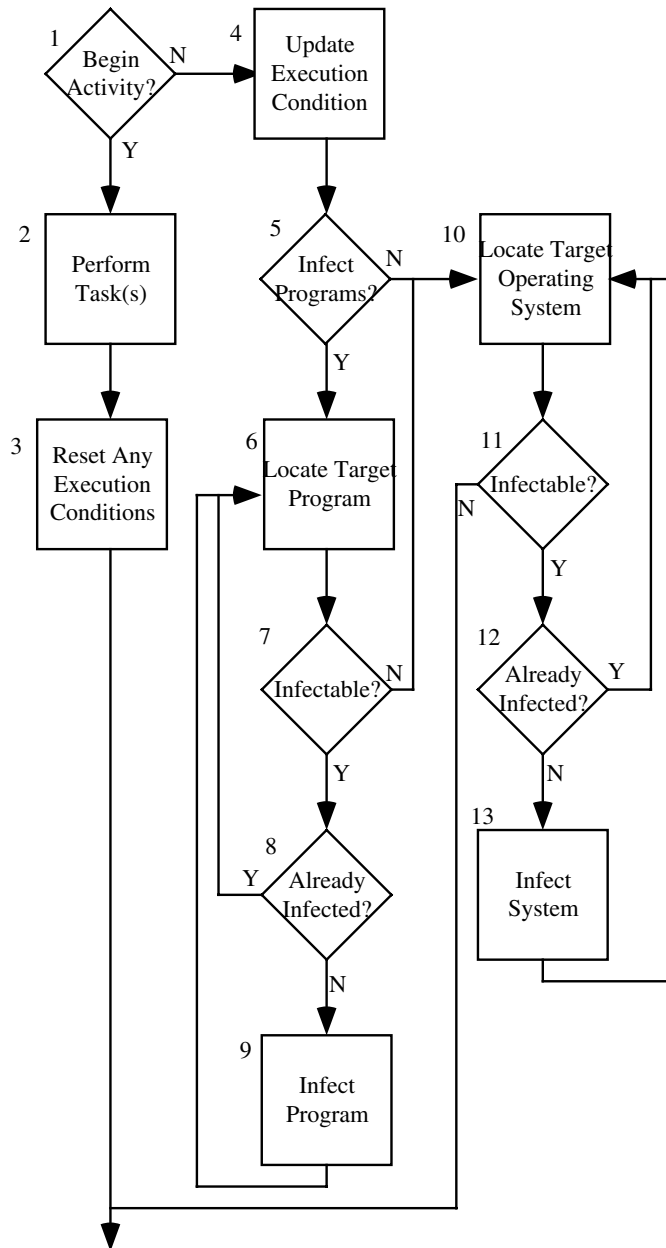


Figure 15.1. A flowchart showing the actions of a virus program.

Step 1: Following activation, the virus will determine if it should execute—if it should carry out its primary purpose. This purpose will be defined by the author of the virus, however it is often file or disk erasure. If the necessary conditions exist for the virus to execute, it will do so and proceed to step 2. If the necessary conditions do not exist, the virus will not execute, but rather adjust any conditions, such as an internal counter, that lead to execution. This would occur, for example, if the virus were to execute on a certain date or after a certain number of disk accesses.

Step 2: After determining that it should begin execution, the virus will proceed to carry out its primary purpose. As previously mentioned, this may be disk erasure or file modification. Other possible activities include temporarily displaying a message or picture, modifying a user's program preferences, or transmitting information to a networked computer. Once the activity has completed, the activation conditions of the virus may need to be changed.

Step 3: Once the virus executes, it may need to reset itself for future execution. If the program or disk was not erased, and the virus still exists it may reset the conditions that initiate its execution. In some cases, the virus may remove itself. After the virus has either reset its execution conditions or removed itself, the virus stops and normal execution begins.

Step 4: Because the virus can not execute, it will update its execution conditions so that future execution will be possible. Examples of information that may be updated are access counts or the number of days since the program was infected.

Step 5: After the virus execution conditions are updated, the virus will attempt to infect other programs. If the virus infects other programs, it will attempt to locate a vulnerable target program. If the virus does not effect programs, but rather operating systems, it will proceed to step 10.

Steps 6 and 7: The virus will attempt to locate a potentially vulnerable program in the system. When a vulnerable program is identified, it is examined to determine if it is already infected.

Step 8: If a program is vulnerable to infection, it may be examined for an existing virus. A virus may infect a program regardless of whether it is already infected or not. To examine a program for infection, the virus looks for a unique signature, or identifier, in the program. The identifier is selected by the author as some value that will only exist if the program is infected. If the identifier is found, the virus assumes the program is infected and will search for another program to infect. In the event that the program is not infected, the virus attaches itself to the program so that it will be activated when the program is started.

Step 9: Viruses infect programs by attaching themselves to the target program and inserting a command that executes the virus. Simple viruses place their code in the program and insert a command at the beginning of the program which causes the computer to execute the virus code. For those familiar with assembly language, this would be a command akin to the jump-to-subroutine command. As the presence of a virus can be detected by an unexplained increase in a program's size, the more intelligent viruses attempt to replace unused parts of a program with the virus code. Thus, the length of the program does not change and the program still becomes infected. This requires a significant understanding of system architecture and program design. In addition to the virus code, a virus identifier must be installed. In many cases, these are one in the same. Some virus authors, however, prefer to use a separate unique identifier to ensure the accuracy of testing for a viruses presence. This is also useful for the advanced viruses whose specific location in a program is not quickly identifiable. Following the installation of the code and identifier, the virus will look for another program to infect.

Steps 10 and 11: As the virus infects operating systems, it will attempt to locate a susceptible operating system. In many environments, the complete operating system is not infected, rather portions of the operating system are infected. For example, the previously mentioned Apple 2 virus only infected the operating system's file manager. Some computers have a start-up, or boot-sector, process that contains information about the operating system that is important when the computer is powered on. Modification of this information or process is also very possible, and has actually become a common means of infecting a computer. Once the virus

finds a susceptible operating system, it must determine if it has already been infected (step 13).

Step 12: A virus will examine the susceptible operating system, searching for an existing virus. Like the activity of a virus in step 8, the virus will look for identifiers that the operating system has already been infected. In the event that the operating system has been infected, the virus will wait until it is in contact with another system and try again (step 10).

Step 13: Immediately proceeding the identification of an uninfected system, the virus will place a copy of itself within the system. The exact location will depend on the author's objectives of distributing the virus. It may attach itself to the file management routines, the user authentication routines, or possibly the system accounting routines. In addition to attaching the virus code and patching the existing files to activate the code, an additional virus identifier may be added to mark the presence of the identifier. This will be used to identify the existence of the virus so that infected systems are not continuously re-infected. As will be discussed below, this can also increase the chance of the virus being found and removed.

The exact steps that a virus undertakes to infect and harm a system will vary with the author's intentions and capabilities. Nonetheless, the general process presented above provides an accurate representation of a virus. Understanding the concepts employed by this algorithm will assist in understanding the general concepts associated with viruses as well as the actions that should be taken to detect, prevent, and remove a virus.

15.1.3 Prevention, Detection, and Removal

The prevalence of malicious code such as viruses require user and system administrators to both protect against becoming infected and to quickly remove any virus that enters the system. Many software products are available to detect, prevent, and remove viruses. These packages, however, are not enough. Because virus detection and prevention often relies on another user's confrontation and identification of a virus, it is possible that existing detection and prevention mechanisms will not identify new viruses. Furthermore, removal is often performed in response to the known presence of a virus, thus, the virus may have already caused irrecoverable damage. Additional action is therefore required to better protect a system and reduce the need for virus

removal. In the event that virus eradication and system recovery are required, steps should be taken to prevent future infection.

15.1.3.1 Prevention and Detection

The first step in protecting a system from a virus is preventing infection. As infection can come from many places, a comprehensive set of protective policies and procedures should be established and adhered to. These policies and procedures should indicate the acceptable sources of software, a list of programs acceptable for use, and a list of actions to perform in the event the presence of a virus is suspected. Some widely accepted procedures are listed in [Table 15.1](#) [15.5, 15.6].

Table 15.1. Suggested virus prevention policies.

- | |
|---|
| <ul style="list-style-type: none">• Use only sealed software that is purchased from a reputable vendor.• Make backup copies of all new software before it is used, and possibly infected.• Purchase and use programs that actively search for viruses and remove them.• Use checksum programs that can be used to identify changes to programs.• Keep logs of all software that is installed and used in the system.• Avoid programs that come from untrustworthy sources such as computer bulletin boards or other on-line services.• Always test new software for viruses before they are used or installed.• Educate users about the danger of viruses and how to prevent infection.• Do not use or install illegally copied software. |
|---|

These techniques will go far in preventing viruses from infecting a system. This is often not enough, however, as viruses are often unknowingly introduced into a system. For example, a co-worker could unknowingly bring a virus into work that their child brought home from school. In this type of situation, quick detection of a virus is the best means of reducing the possibility of harm or damage. Detection is accomplished using any one or more of the following four techniques: (1) user identification of symptoms, (2) virus signature identification, (3) checksum verification, and (4) access monitors [15.3, 15.6]. As will be discussed, all of these techniques, except the first, are readily available to users in commercial anti-virus

software. A fifth detection technique not listed above is identifying the aftermath of a virus; hopefully, this method will rarely be employed.

The original, and still common, means of suspecting or detecting a virus is by users identifying symptoms of infection. Common symptoms include slower processing, a change in the system time, altered file attributes, and other erratic system behavior. Another more common side effects is an increase in the size of a program [15.3]. Some of the more knowledgeable virus authors are aware of these symptoms and program as to avoid them. They are able to write efficient virus code that does not modify the length of a program, does not negatively impact processing speed, and does not cause excessive disk access. Preventing these symptoms, however, will not keep a knowledgeable and prepared user from detecting a virus. When a user suspects a virus has infected the system and no single symptom seems to confirm this, a virus detection program may provide some assistance.

The most common means of detecting a virus is with the aid of commercial software, such as McAfee, Norton Anti-Virus, or many of the other widely available packages. These programs scan a system and identify known viruses by searching for specific information about the virus. This information is another form of virus signature, however, it may not be identical to that used by a virus to prevent re-infection. In most cases, this scanning is sufficient to identify the presence of a known virus. As not every virus is well document and known to these programs, a new virus may go unnoticed. Most of the software vendors address this issue by offering free virus definition updates on a periodic basis. When added to the anti-virus software they detect and remove newly identified viruses. Because new viruses are being created every day, it is important for users of this software to check with their vendors frequently.

Another means of providing protection is to employ checksum calculating programs. These programs examine a data file or application and calculate a unique value, or checksum, based on the information being examined. When checksums of the same information taken at different times are compared, they should match. In the event that the two checksum values do not match, the data file or application has been modified. If modification has not been authorized, this may indicate that the file has been infected. While this process is very effective at identifying questionable modification, it can require significant storage space and user time. Fortunately, the more popular virus detection programs provide similar processes that require little more user assistance than configuration.

Another technique often used in detecting viruses employs access monitors. Access monitors are programs that examine all disk access looking for suspicious behavior. Suspicious behavior includes modification to normally unaltered system files of programs. In addition to identifying the unwanted accesses, most monitors inform the user of the activity and prevent it from completing. This is such a useful process that most of the existing detection software includes some form of monitor. In many cases the monitor is limited to watching the activity associated with specific files as

opposed to the entire system. These monitors warn users of questionable file access and allow the user to determine the course of action—allow, prevent, or always allow.

15.1.3.2 Disinfection

Once the presence of a virus has been confirmed, the entire system must be disinfected; the virus must be removed. Disinfection does not just mean that a new copy of an infected program should be installed; other infected files will simply reinfect the program. Proper virus removal includes a comprehensive virus search and removal on both fixed media (i.e., hard drives) and removable media such as floppy disks, removable cartridges, and backup tapes. Examination of all fixed media at one time will work to ensure that the virus is completely removed from the system. Examination of all removable and backup media will prevent a user from reinfecting an otherwise uninfected system and beginning the infection process anew. Care should also be made to examine any user created CD's to verify that their contents will not be responsible for continuous infection. Many commercially available programs assist in the detection and removal process. It is suggested that these programs be used to remove viruses as they will be able to both locate every copy of the virus, and completely remove any trace of the virus [15.9]. Furthermore, a virus may be too well imbedded into a program or operating system for manual removal to be either complete or accurate.

Automated virus removal programs will scan all programs and operating systems for virus signatures or other virus identifying code. Once found, these programs modify the virus carrier by removing the virus code and activation statements. Upon completion of this process all known viruses will be removed from the system. Great care must be taken during this process as not to reinfect the programs or systems. It is not unusual for people to disinfect with an infected disinfection program. While all traces of the virus will appear to have been found and removed, the disinfection program will reinfect the system.

Commercially available disinfectant programs are not the only means of removing a virus. Another, more drastic, method is to completely erase all media and reinstall the system software and programs from their originals. This method is time consuming and often impractical or impossible. Furthermore, If the replacement software is also infected, reinfection is almost certain.

15.1.4 Special Viruses

There are three viruses that deserve special attention: *the Good Times virus*, *the Macro virus*, and *the Boot Sector virus*. They are specifically mentioned because they act outside the norm for a virus and they have become very prevalent in

recent months. The Good Times virus is worth special mention because no other virus has attracted so much attention and been so harmless. The Macro virus is discussed because it has clouded the issues of how a virus works and what type of information it attaches itself to. Finally, the Boot Sector Virus is mentioned because it is an excellent example of a virus that effects the system as opposed to a program.

15.1.4.1 The “Good Times” Virus

Starting around December, 1994 a people on the Internet started receiving e-mail about a new virus, the Good Times virus. These messages warned e-mail users about a new e-mail message based virus that, if read, would destroy the contents of the local hard drive and force the computer processor to work so much that it will destroy the computer. The text of one of email warnings is provided in [Figure 15.2](#). As can be expected, such a disastrous virus would result in multiple similar warnings being sent throughout the Internet. Some specifically request the reader to forward the message to friends and other system users. Other warnings indicate that the virus will automatically forward itself to any email addresses found in sent or received email. All of them, however, were a hoax.

The FCC released a warning last Wednesday concerning a matter of major importance to any regular user of the InterNet. Apparently, a new computer virus has been engineered by a user of America Online that is unparalleled in its destructive capability. Other, more well-known viruses such as Stoned, Airwolf, and Michaelangelo pale in comparison to the prospects of this newest creation by a warped mentality.

What makes this virus so terrifying, said the FCC, is the fact that no program needs to be exchanged for a new computer to be infected. It can be spread through the existing e-mail systems of the InterNet. Once a computer is infected, one of several things can happen. If the computer contains a hard drive, that will most likely be destroyed. If the program is not stopped, the computer's processor will be placed in an nth-complexity infinite binary loop - which can severely damage the processor if left running that way too long. Unfortunately, most novice computer users will not realize what is happening until it is far too late.

Figure 15.2. One of many Good Times virus warnings.

The good news is that the Good Time virus does not exist as described in the warning. The bad news is that the warning was repeatedly sent to millions of people a

day. Numerous large companies fell prey to the hoax with administrators and executives e-mailing users to be careful. The U.S. government also fell victim to the virus. The virus also made its way overseas where it was translated into multiple languages. The virus was able to gain this popularity largely because the victims were unaware of how a virus worked and the actual damage it could cause. This leads to a very good questions, *is this type of virus possible?*

There has been great debate amongst virus enthusiasts as to whether an e-mail virus, like the one described in the warning, is possible. Recall that a virus is a small piece of code that requires a host program to execute. The e-mail is simply a piece of data that is read, not executed. For the virus to behave the way it is described in the warnings, it would have to have an attached file and the e-mail program would have to be configured to automatically run the attached program. The activity that these warnings describe is more closely related to a Trojan horse than a virus. Even if this type of virus were possible, and some day it may very well be, the virus would not be able to do the damage the warning suggests.

Since the virus was released two significant events have occurred. First, many copy-cat hoax viruses have appeared. These viruses talk about get rich quick schemes, a false FCC modem tax, e-mail from pen pals, e-mail that notifies people when their outgoing message was delivered, and e-mail hoaxes that are attributed to publicity stunts. These second event is the release of a real virus called “Good Times.” Anti-virus professionals have identified it to be similar to existing viruses and have since called it the “GT Spoof”, or Good Time Spoof, virus.

Because of the large number of virus hoaxes that exist, it is important for administrators and recipients of such warnings to verify the authenticity of the warning. There are many excellent virus hoax listings on the Internet, two of the more popular are: <http://www.stiller.com/hoax.htm> and <http://www.kumite.com/myths>. These sites are also the source for excellent information regarding real viruses and the latest releases.

15.1.4.2 The Macro Virus

The presence of the Macro Virus as grown dramatically since it was first identified. This is a result of several factors. First is the ease in which a macro virus can be created. Second is the wide spread use of programs that support macros. Third is the uniqueness in which the virus exists and propagates. But most significant to the macro viruses rampant growth is the fact that it is the first cross platform virus to exist. Before these issues are discussed, however, an understand of the Macro virus is necessary.

A **macro** is a small segment of code that is typically used to simplify complex or redundant tasks in application programs such as Microsoft Word or Microsoft Excel.

Common uses are to perform detailed file formatting or complex mathematical calculations. Macros can also be created to perform basic file access operations to read and write files, as well as operating system functions that can create other programs or modify the system settings. It is this type of activity that a macro virus performs.

Macros are also designed to be small and portable. They can either exist as part of a document or with a library of macros associated with a program. The choice of where the macro is stored, either with a document or with the library, is up to the user; but modifiable by the macro. Macros can also be configured to require a user to manually start it or automatically execute when the document is opened or the program begins. It is with these options that the macro virus operates and propagates

The macro virus begins as part of a document, maybe a help file or a joke file or real data. The macro is designed to perform some annoying task such as change the screen saver or only allow documents to be saved in one format; or delete key system files or the last opened data files. The macro is also programmed to do three key tasks: (1) begin execution automatically, (2) copy itself to the library if it is part of a file, and (3) copy itself to all open files if it is in the library. When the file containing the macro is introduced to the program, the program executes the macro which in turn places a copy of itself in the program library and executes its primary task. As each new document is loaded, the macro is copied into it.

The macro virus has gained in popularity for many reasons. First, the macro language is often made simple so that many non-programmers can take advantage of the benefits of macros. In many cases, the macro language is a derivation of the BASIC programming language. Second, some of the most popular programs have macro capabilities and can therefore be involved in the replication and execution process. The macro language is often so widely used that a macro written for one program may work in another with little or no modification. Third, many people believe that the macro cannot be passed via documents; only by programs. The macro virus is unique in that the program is required to execute the macro, but the data file is the transport mechanism. Nonetheless, the macro virus is still a virus and thus requires a host program to execute. The fourth reason the macro virus is so popular is that it is platform independent; the same identical macro can run on a Macintosh and an IBM Compatible PC. This is very unique as the two systems are very different and have had no other virus in common in this identical manner. The end result of these issues is that almost anyone can write a simple macro, distribute it to a wide audience easily, and have it function regardless of the end user's computer. This of course can lead to some exceptionally dangerous situations. Fortunately, the latest macro running programs and anti-virus utilities can detect and disable the macro viruses.

15.1.4.3 The Boot Sector Virus

The Boot Sector virus, or Boot Record virus, infects the startup process of IBM compatible PC's. Specifically, the virus infects the system's Master Boot Record (MBR) or the active partition's boot record. The virus then remains in memory and duplicates itself onto any writeable disks with a boot record. If this happens to a floppy disk or other removable media, the disk or media becomes a carrier that will infect other operating systems.

The virus operates by taking control of the system after the system is turned on or reset. When the system boots, the BIOS loads the boot record of the infected system into memory. At this time, the system gives complete control of the system to the virus. The virus then reserves conventional memory and effectively tricks the computer into believing that the memory containing the virus is unavailable to the system. The virus is then able to stay resident without the system being able to use or modify the virus resident memory. The virus typically attempts to install itself as a memory resident driver so it can monitor all future system activity and infect other copies of the operating system it encounters. Once the virus has safely installed itself in the system, it returns normal operations to the system and loads the original boot record to continue the startup process.

The potential damage that can result from a Boot Sector virus ranges from file loss from writing the original boot record over data to total system crash when the user performs some action that inadvertently writes over the good boot record. Some users that employ special programs that make multiple operating systems available on startup may have to completely reinstall one or more of their systems.

15.2 Worms

A worm is very much like a virus in that it replicates itself and attacks a system with the potential to do irrecoverable damage. Unlike a virus, a worm is a stand-alone program that infects a computer system and infects other computers only through network connections [15.1, 15.5]. Once a worm infects a system, it actively seeks out connections to other computers and copies itself onto these systems. In addition to propagating from computer system to computer system, worms often perform malicious activities. A worm's activity is not limited to just modification or deletion of files. Since the computers are connected via a computer network, the worm can communicate information back to the author regarding such things as user passwords, network service information, and even proprietary research or information. Furthermore, a worm may be able to completely disrupt normal operations on a computer, thus causing a denial of service attack. This often occurs when a worm does not check a

system to see if it has already been infected and multiple worm programs execute on one computer system. To understand how this could happen, it is useful to know how a worm infects a system.

15.2.1 Infection

Unlike creating a virus, creating a worm is a more difficult task. For a worm to properly function, the author must be knowledgeable with communication protocols, network system vulnerabilities, and operating system details such as file locations, file contents, and file manipulation [15.3, 15.5]. As such, there are significantly fewer occurrences of worms than of viruses.

For an operating system to become infected, a worm must either be initially released into the computer, or have migrated over a computer network [15.5]. While these are the only methods of infection, the many possible network sources of a worm make them difficult to fend off. One means of preventing infection is to identify and control any security vulnerabilities or holes in a system. This too is difficult, as many administrators are not aware of security vulnerabilities until they have been exploited. Because many of these security deficiencies involve either easily determined user passwords or uncontrolled network services worms are capable of easily infecting a system [15.1].

Once a worm has gained access to a single system, it is quite easy to gain access to other systems. By taking advantage of trusted host relationships (e.g., UNIX .rhosts files), a worm may be capable of quickly infecting numerous systems without painstakingly attempting to exploit a system's security vulnerabilities. In the event that trusted host lists are unavailable, many worms will attempt to penetrate a system by guessing user passwords [15.1]. Previous experience has demonstrated the high success rate of password guessing [15.2, 15.7]. When both password guessing and trusted host accessing fails, a worm may attempt to exploit known security holes. This technique requires a worm's author to be very familiar with the inner-workings of a computer's network services; how network services work, and how they may be exploited. An example of one such incident where this knowledge was used is the widely known Internet Worm¹. Details about how this and other system penetration methods are used in distributing a worm are discussed below.

15.2.2 Theory of Worms

The process a worm undergoes when infecting and attacking a system is identified in [Figure 15.3](#). The flowchart in [Figure 15.3](#) lists the logical actions a worm takes to

¹ This incident is discussed in detail in chapter 15.2.

achieve its goals. Again, the numeric labels attached to each step of the process do not indicate the order in which the steps are undertaken; they are provided only as a means of providing detailed descriptions.

Step 1: A worm must first enter the system. In the event that the worm currently resides in the system, the new worm should terminate. This step is for worm installation that does not occur through network communication channels. This step will be performed for all initial placement of a worm by a user as opposed to a networked computer system. Before a worm is installed by network communication means, step 4 will ensure that a duplicate worm is not installed.

Step 2: After the worm has copied itself into the system, it will begin its execution. To reduce the chance of the worm being noticed and removed, it may attempt to hide its presence by masquerading as a system task. When the task is firmly installed into the system, it will begin its attempt at compromising other computer systems. One method worms use to enter and infect other systems is to exploit files that allow users to enter other systems without authentication (e.g., UNIX .rhosts files). These files allow worms, as well as users, to enter other systems without being challenged for authentication. A second method of gaining access to remote systems is by guessing user passwords. As many users have passwords that are easy to guess, this can often provide the necessary means to enter a system. This is also effective because many administrators do not remove, or otherwise secure, system default passwords, thus allowing anyone to gain administrative privileges. A third means of gaining access to another system is by exploiting known security holes. One well-known worm, for example, exploited holes in a computer's network services that allowed the worm to execute programs on the remote system. In a similar manner, some poorly administered systems provide utilities that outright allow remote systems to execute programs. Regardless of how a worm is capable of executing programs on remote systems, the end result will be the installation of the worm.

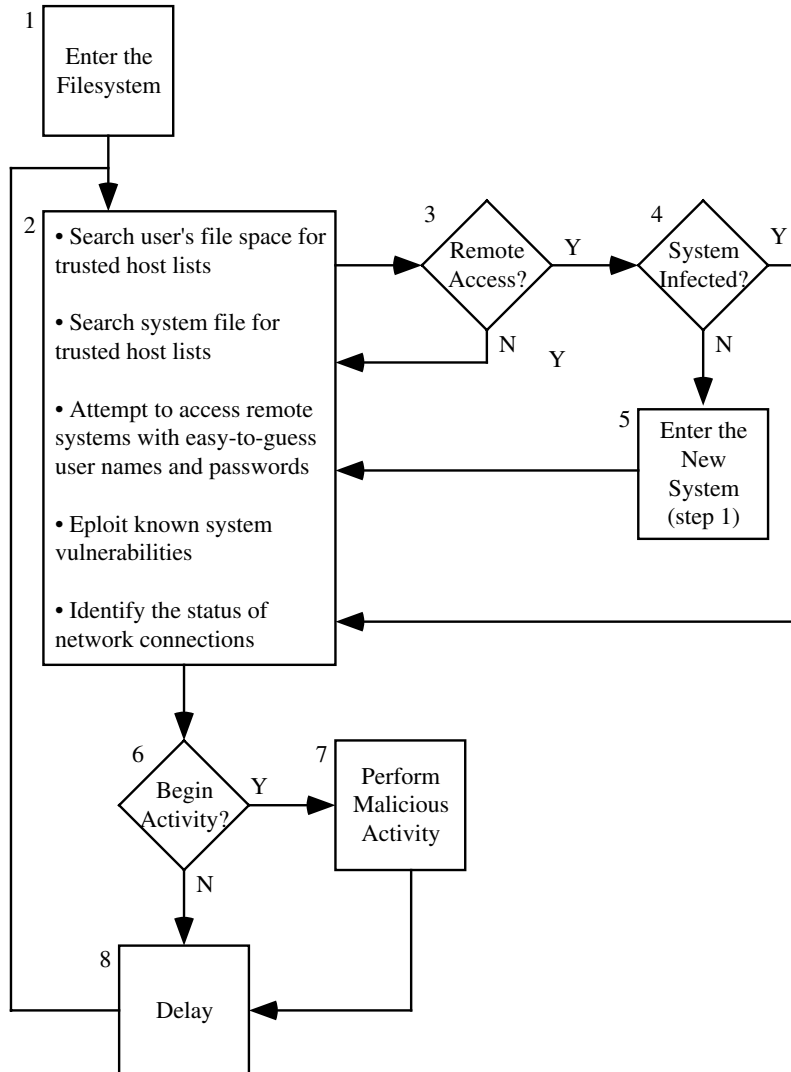


Figure 15.3. A flowchart identifying the actions a worm undertakes.

Step 3: Once a new system and means of attack have been identified, the worm will begin its attack on the system. If the worm is unsuccessful at accessing the system, it will select another means of attack. In the event the worm is successful, it will attempt to determine if the newly accessed system is already infected.

Step 4: After a worm gains access to another computer system, it will attempt to determine whether the system is already infected. If the system is not infected, it will install itself. If the system was previously infected the worm will attempt to gain access to another system. In the event that the worm infects the remote system before determining if it was previously infected, multiple copies of the worm will infect the system. This will eventually lead to a denial of service attack as the system will be too busy supporting the multiple worms to perform its normal operations². Clearly, this will be noticeable to the system's users and administrators.

Step 5: Upon determining that a system has not been infected, the worm will install itself on the remote system. Once the worm is installed it will begin to seek out additional systems to infect. The installation process will be similar to that described in step 2—the worm will attempt to conceal its presence by masquerading as a system program and then begin its attack on other systems.

Step 6: After a given number of access attempts (as selected by the worm's author) the worm will determine if it should perform its intended action. This action may be performed immediately after a worm has gained access to a system, after a certain event occurs, or possibly after a specific amount of time has elapsed. Again, the exact triggering conditions will be programmed by the worm's author. For example, if the author desires the worm to steal user passwords, it may be programmed to wait until a certain number of passwords are captured before it sends them to the author. If the worm's purpose is to display a new years greeting, the worm may be programmed to activate on January 1.

Step 7: Not every worm has a malicious goal or purpose. Those that do, however, will want to perform that activity. Possible activities include stealing user passwords, stealing proprietary information, displaying a simple message, erasing files, or bringing a computer system to a halt. The final action in this step may be for the worm to remove itself. For example, a worm programmed to display a new years message may be programmed to remove itself on

² This mistake led the Internet community to identify the presence of the Internet Worm (see chapter 15.2).

January 2. If this is the case, the worm may terminate its processing and no longer attempt to infect other systems.

Step 8: A worm that continuously processes will raise suspicion amongst system administrators and user. To reduce its chances being noticed, a worm may periodically become dormant and delay its operations. After it has completed its sleep cycle, it will continue to search for susceptible systems.

The exact execution steps that a worm will follow may vary slightly from those described above. Regardless of the minor differences that may exist, the means by which an infection is prevented and a worm is removed will be similar. This similarity stems from the fact that a worm's communications channels, regardless of their format, must be closed.

15.2.3 Prevention and Removal

After the presence of a worm has been detected in a system, the administrator will work to remove it. This can be a difficult endeavor as the worm may be successfully reinstalled by a neighboring computer. This task may require removing systems from the network to prevent reinfection. It may also require modification to an organization's external network connections to identify the worm and prevent it from entering the network.

15.2.3.1 Preventing Worm Attacks

Preventing the infection of a computer system from worms is similar to protecting a system from an intruder. Both types of attacks attempt to exploit the same system vulnerabilities to gain access. The primary difference between the two types of attacks is the deterministic methodology employed by a worm. As a worm can only perform the actions selected by its author, in the manner in which they are programmed, its modus operandi is unlikely to change. Knowing the methods employed by a worm, as well as the vulnerabilities the worm attempts to exploit, an administrator can protect a system from infection. Furthermore, this information will be required when worm removal is performed.

As with intrusion prevention, methods such as firewalls and access control can significantly reduce the means by which a worm can enter a system. Three general security steps an administrator can take towards preventing infection are: (1) periodic security reviews of systems, (2) periodic examination of user account passwords to

ensure their strength under attack, and (3) disallow the use of system files that grant trusted remote access without the proper authentication.

15.2.3.2 Worm Removal and System Recovery

Worm removal is a difficult task. Neighboring computer systems will immediately work to reinfect a clean, or uninfected, system. There are three possible techniques for preventing a system from becoming reinfect: simultaneous wide-spread removal, system isolation, and infection prevention coupled with local worm removal. Each of these techniques will allow a worm to be removed from a system without being reinfect, however, the viability, advantages, and disadvantages of each technique vary greatly.

The first technique, simultaneous widespread removal, is the process of removing the worm from every system at approximately the same time. The times at which each copy of the worm is removed does not need to be exact; it need only be close enough to prevent a worm on one system from completing a processing cycle and successfully accessing and infecting another system. In a small network, such as a network with ten computers, this may be possible. When a network has a larger number of computers or is attached to a larger network, this process is no longer viable. Because the removal activity of a smaller network can be more greatly controlled, every system can be cleaned simultaneously. Larger networks, such as the Internet, do not provide this type of control. If just one infected computer on the network is not cleaned at the same time as the others, the worm will reinfect the network. In the event it were possible to clean every system at the same time, network propagation delays would work against this process and the system may still remain infected. Simply cleansing every infect computer simultaneously is not a reasonable solution for every computing environment.

The second technique for preventing reinfection is system isolation. By either physically or logically separating a system from a larger network, the isolated system may be easier to cleanse without reinfection. This solution has received much debate [15.2, 15.4]. Some researchers argue that isolation will allow a site to cleanse itself without fear of reinfection. They further suggest that this will prevent a local worm from infecting other systems [15.4]. Other analysis indicates that remaining attached to other networks provides significant advantages. Isolation may prevent the timely exchange of security information and patches. Furthermore, isolation of systems that normally act as data and mail relays will delay the timely flow of other information to other systems.

The third, and most successful, technique in preventing reinfection is to apply local worm removal methods. That is, patch all of the holes exploited by a worm and then remove any existing copies. Because the worm will be unable to utilize any

previously identified security holes, it will have to either use another security hole or not fail to propagate. As a worm's actions are limited to only those included by the its author, there will be point at which every security hole utilized by a worm will be patched. Once this occurs, the worm can be removed without fear of reinfection. This process may require administrators to remove some of the functionality of their system or lock accounts with easily guessed passwords. These actions, however, are both temporary and easily worked around.

15.3 Trojan Horses

Trojan horse programs were named because of their functional similarity to the mythical horse of Troy. Trojan horse programs are advertised to perform one function while, in fact, they perform a different function. This alternate function often performs a covert action such as steal user passwords. While the alternate function always executes in some manner, the advertised functionality may not necessarily exist. For example, a program may be advertised as the greatest game ever made, but merely erase a disk and halt. Trojan horse programs that wish to function in a concealed manner, however, will perform their advertised task as not to arouse suspicion. A common example of this is a system's user login program that not only authenticates users, but records a user's plain text password for unauthorized access in future.

As will become increasingly obvious throughout this section, Trojan horses are very similar to viruses; both infect programs or operating systems; both hide their presence behind another program; both hide their activity behind another program. This similarity has often caused confusion concerning the relationship between Trojan horses and viruses and the appropriate classification of malicious code [15.6, 15.8]. While the two are similar, their differences are significant. The differences between Trojan horses and viruses are twofold: (1) Trojan horses actively advertise an alternate functionality to bait the unsuspecting user, and (2) viruses attempt to attach themselves multiple host programs. These differences are enough to warrant somewhat different approaches to detection, prevention, and recovery.

15.3.1 Receiving Trojan Horses

The means by which a Trojan horse can enter a system are not as great as that of either worms or viruses. As Trojan horses are neither self-replicating nor self-propagating, user assistance is required for infection. This occurs by users installing and executing programs that are infected with a Trojan horse. These programs may come from many possible locations. Three popular sources for programs infected with Trojan horses are bulletin board systems, public access file servers and Internet sites,

and computer labs such as those found at universities [15.6]. These locations are often used because they provide a great opportunity for wide spread release of a Trojan horse. Furthermore, the files found on these systems—public domain, shareware, and possibly pirated software—come with little or no guaranty of any sort. That is, the person downloading or using the software does so at their own risk.

The placement of Trojan horses in public domain software, shareware, or pirated software is very strategic. These types of software are popular to the public as they are available for little or no money, albeit sometimes illegal. Regardless of their legality, this popularity makes them excellent candidates for achieving the desired effect of widespread distribution. The additional benefit of using public domain software or shareware is that the Trojan horse can be interwoven directly into a program as opposed to being added to an existing program. The perceived benefit of infecting pirated software is that it may reach the many unethical users who traffic pirated software.

Upon activation, a Trojan horse will perform its programmed task. As previously mentioned this task can be as trivial as adding bogus files to a system, or as severe as stealing passwords or erasing data. Trojan horses do not, however, replicate themselves and infect other programs—this type of activity would indicate the presence of a virus as opposed to a Trojan horse.

15.3.2 Theory of Trojan Horses

The similarity between Trojan horse and viruses will again be noticeable when comparing the algorithms of the two. The Trojan horse algorithm depicted in [Figure 15.4](#), is nearly identical to the algorithm for viruses in [Figure 15.1](#). The key difference between the two is that Trojan horses do not perform the steps that lead to infecting other programs or systems.

Like a virus, once a Trojan horse infected program is executed or an infected operating system performs a task, the Trojan horse will execute. The Trojan horse will first determine whether it should execute. If the Trojan horse meets all of its internal conditions to execute, it will do so. Following completion of the task, the Trojan horse may either reset its conditions or remove itself. After the Trojan horse has adjusted its execution conditions, it will complete and allow normal computation to continue. With the assistance of [Figure 15.4](#), the steps involved in this process are detailed below.

Step 1: Upon activation, the Trojan horse will determine if it should execute—if it should carry out its primary purpose. This purpose will be defined by the author of the Trojan horse. If the necessary conditions for execution exist, the Trojan horse will do so.

In most cases, there are no specific conditions and the Trojan horse will run. If the necessary conditions do not exist, the Trojan horse will not execute, but rather adjust the conditions that lead to execution.

Step 2: Upon determining that it should begin execution, the Trojan horse will proceed to carryout its primary task. This is often unauthorized password or file theft. Once the activity has completed, the activation conditions of the Trojan horse may need to be changed.

Step 3: Once the Trojan horse has executed, it may need to reset itself for future execution. After the Trojan horse has reset its execution conditions, the Trojan horse stops. If the Trojan horse is part of another program, the program typically executes.

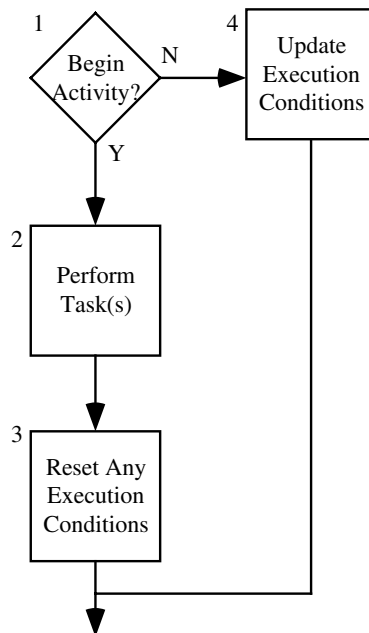


Figure 15.4. A flowchart identifying the general actions of a Trojan horse.

Step 4: Because the Trojan horse does not meet the necessary conditions to execute, it will only update its execution conditions.

Examples of information that may be updated are access counts or the date a file or password was last stolen. Again, like viruses, this will depend upon the authors desired effect.

The precise execution steps of any given Trojan horse may vary from those presented here. This variation, however, will be minimal and the general algorithm presented above will accurately represent the activity of a Trojan horse. Furthermore, this difference will not effect the means by which a Trojan horse is detected, and removed.

15.3.3 Prevention, Detection, and Removal

Unlike viruses or worms, the prevention and detection of Trojan horses can be largely accomplished by educating the users. Because Trojan horses neither propagate nor replicate, they must come from programs that a user introduces to a system. If the users can be educated to carefully screen and examine programs before they are used, the introduction of a Trojan horse can be severely reduced. Not every Trojan horse, however, will be identified before it enters a system. Simple methods do exist to reduce this threat and eventually lead to the removal of Trojan horses [15.6].

15.3.3.1 Trojan Horse Prevention and Detection

As with virus prevention, Trojan horse prevention can be largely accomplished by enforcing a strict software installation policy and educating users. By preventing users from installing software from questionable sources and unauthorized software, the possibility of infection is greatly reduced. By installing only new and original copies of commercially available software, the paths of Trojan horse infection are blocked. Because it is difficult to control every user and prevent them from installing unauthorized software, users must be educated in the dangers of Trojan horses. Because users often under estimate the possibility of infection and dangers that are associated with Trojan horses, they ignore security policies and procedures. Education should reduce this problem and prevent users from carelessly installing software.

As infection prevention is not always successful, Trojan horse detection techniques should be used. Three techniques available to detect Trojan horses are observation, checksums, and audit trail analysis. Observation, the most commonly used technique, is performed by administrators in an informal manner. That is, many administrators do not actively search for Trojan horse, rather daily system use, modification, and examination are used to detect Trojan horses. As observation is not the most scientific or exacting means of detecting Trojan horses, other methods such as checksums and audit trails should therefore be used.

As with detecting viruses, checksums can be used to detect the presence of Trojan horses. As current checksum technology severely reduces the possibility that a file can be modified and retain the old checksum values, is considered a reliable and effective means of Trojan horse detection.

Audit trails can be used to identify the activity taken by intruders and possibly Trojan horses. Under most circumstances, a Trojan horse's activity will not be recorded by the audit trails. If, however, the trails do identify the unusual activities of a Trojan horse, its presence can be detected. This will require a detailed analysis of the audit trails to identify the program infected with a Trojan horse. Assuming this process locates the Trojan horse, the Trojan horse can be removed. Because the audit trails may not be detailed enough to identify the infected program, a security administrator may not be able to locate the Trojan horse. If the presence of a Trojan horse is suspected, but it cannot be located, an administrator will have to rely on observation and checksums or search for the questionable file.

15.3.3.2 Trojan Horse Removal

Because Trojan horse programs do not replicate themselves, Trojan horse removal is easier than either virus or worm removal. The first step of the Trojan horse removal process is locating the Trojan horse. Removal simply requires replacing the file with an unaltered, trusted copy of the program. This replacement, however, should not be the primary concern of an administrator. Of greater concern is the activity of the Trojan horse. If the Trojan horse program captured and sent out user account and password information, a significant problem still remains. Thus, system security relies upon determining the Trojan horse's actions and intentions. After an administrator has determined the purpose of a Trojan horse, damage response can begin.

15.4 Summary

Programs that intentionally modify or destroy system and user files are called malicious programs, or malicious code. Three examples of malicious code are viruses, worms, and Trojan horses. The programs propagate from one computer system to another by either attaching themselves to executable programs that are copied by users or by migrating over a computer network. Once inside a computer system they corrupt data, erase files, or steal sensitive information for their author.

The probability of being victimized by malicious code can be largely reduced through the enforcement of an anti-virus and general software installation policies and procedures, and by user education. The software installation procedures should indicate the means by which a user can install software into a system. The procedures should

also dictate the acceptable sources for software packages and how software should be examined and tested before it is permanently installed. User education will also assist in preventing attack as informed users will understand the ramifications of installing malicious code.

In addition to software installation policies and procedures, and user education, specialized tools are available to prevent and detect the presence of malicious code. These programs periodically examine systems and files for the signs of malicious code and report their findings to a system administrator. After a malicious program has been removed, its actions and intentions should be identified so that additional actions can be taken to counter its actions.

15.5 Projects

- 15.1 Compare the three types of malicious code that were discussed in this chapter. What discerning characteristics does each have? Are there other types of malicious code that were not discussed? Draw a flowchart describing the general actions for each of these other types of malicious codes. How do these other types of code compare to the virus, worm, and Trojan horse?
- 15.2 How could malicious code be used in a non-malicious way? Describe how each of the three types of malicious code, virus, worm, and Trojan horse—could be used to benefit a computer system or network. What are the potential hazards or disadvantages of such an approach?

15.6 References

- 15.1 Curry, David, UNIX System Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1992.
- 15.2 Eichin, Mark W. and Rochlis, Jon A., “With Microscope and Tweezers: An analysis of the Internet Virus of November 1988”, *Massachusetts Institute of Technology Technical Report*, Massachusetts Institute of Technology, February 1989.
- 15.3 Fites, Philip; Johnston, Peter and Kratz, Martin, The Computer Virus Crisis, Van Nostrand Reinhold, New York, New York, 1989.

- 15.4 Garfinkel, Simon and Spafford, Gene, Practical UNIX Security, O'Reilly & Associates Inc., Sebastopol, California, 1992.
- 15.5 Russell, Deborah and Gangemi Sr., G. T., Computer Security Basics, O'Reilly & Associates Inc., Sebastopol, California, 1991.
- 15.6 Shaffer, Steven L. and Simon, Alan R., Network Security, Academic Press, Inc., Cambridge Massachusetts, 1994.
- 15.7 Stallings, William, Network and Internetwork Security: Principles and Practices, Prentice Hall, Englewood Cliffs, New Jersey, 1995.
- 15.8 Stubb, Brad and Hoffman, Lance J., "Mapping the Virus Battlefield: An Overview of Personal Computer Vulnerabilities to Virus Attack", *Institute for Information Science and Technology Report GWU-IIST-89-23*, George Washington University, August 1989.
- 15.9 Wack, John P. and Carnahan, Lisa J., "Computer Viruses and Related Threats: A Management Guide", *NIST Special Publication 500-166*, National Institute of Standards and Technology, August 1989.

15.7 Extended Bibliography

- 15.10 Foster, Edward, "Virus Stories Can Sell Papers but may contain a Trojan Horse", *InfoWorld*, Vol. 12, No. 14, April 2, 1990, p. 41.
- 15.11 Gibson, Steve, "Computer Viruses Follow Clever Paths to evade Detection", *InfoWorld*, Vol. 14, No. 8, February 18, 1991, p. 28.
- 15.12 Robertson, Wayne, "The Best defense Against viruses my be sheer luck", *Network World*, Vol 7, No. 34, August 14, 1990, p. 28.
- 15.13 Stephenson, Peter, "A Reality Check on Virus Vulnerability", *Lan Times*, Vol. 5, No. 6, March 22, 1993, p. 57.
- 15.14 Stoll, Cliff, "An Epidemology of Viruses & Network Worms", *Proceedings of the 12th Annual National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 369-377.

16

SECURITY STANDARDS

In an attempt to attain a consistently high level of computer security, several government sponsored organizations have established their own computer security standards. The standards are used to determine the security classification that a hardware or software product is assigned. The standards identify the security criteria that a product must meet in order to be considered for use by many governmental departments and some private organizations. This classification is based upon the security features implemented and employed in a product. Naturally, each government can be expected to have their own security needs and therefore their own security standards. Until a universal criteria is released, there are at least three major standards developed and used throughout the world. These standards are: (1) the United State's Department of Defense Trusted Computer System Evaluation Criteria (TCSEC), (2) the Communications Security Establishments (formerly known as the Canadian System Security Centre) Canadian Trusted Computer Product Evaluation Criteria (CTCPEC), and (3) the joint France, Germany, Netherlands, and United Kingdom Information Technology Security Evaluation Criteria (ITSEC). There are many less formal standards which are employed by other governments and organizations. Some of these have become outdated and replaced with one of the above mentioned standards, thus, reducing the number of existing security standards to a small handful.

In an effort to further reduce the number of existing security standards, a single unified standard is being developed. This standard is called the Common Criteria (CC). The CC encompasses all of the requirements of the previous standards without invalidating the existing product security classifications. The CC, as well as each of the standards that are going into the development of the CC, are discussed below. Before this discussion is presented, however, a brief history of the various standards is necessary. [Figure 16.1](#) identifies the relationships, in both time and content, of the standards.

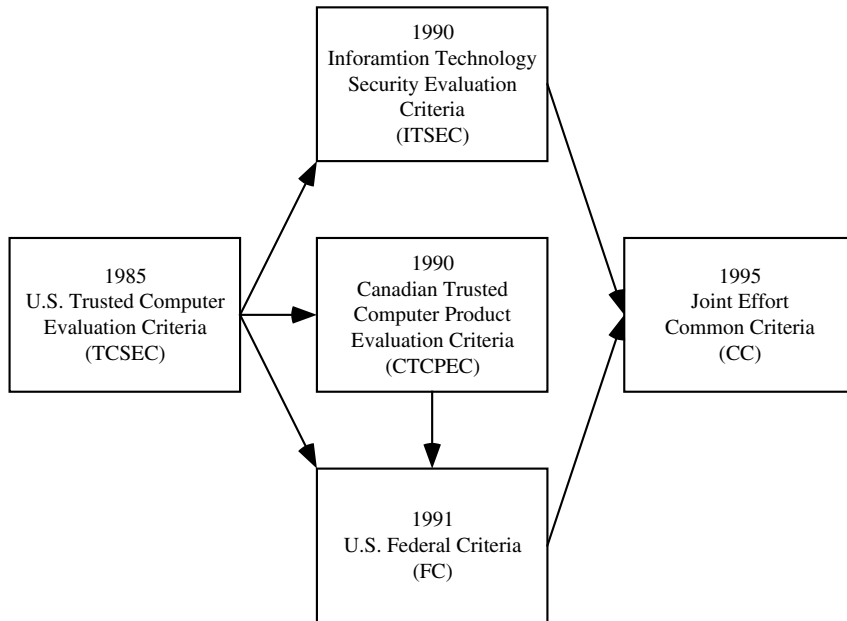


Figure 16.1. The evolution of security criteria.

16.1 The History of Security Standards

The United States Defense Science Board published a report in 1970 containing technical recommendations on reducing the threat of compromise of classified information processed on remote-access computer systems [16.9]. Work stemming from this report led to the creation of the Department of Defense's (DOD) first computer security standard, the Trusted Computer System Evaluation Criteria (TCSEC). This criteria was first published in December 1985 in accordance with DOD Directive 5200.28, "Security Requirements for Automatic Data Processing (ADP) Systems" [16.9]. The purpose of this document was to provide technical security criteria and evaluation methodologies for the support of an ADP system security policy. The value of this document was not only seen in the United States, but overseas as well.

In 1990, a group of four governmental bodies from France, Germany, the Netherlands, and the United Kingdom gathered to create the first draft of a similar document, the Information Technology Security Evaluation Criteria (ITSEC). The ITSEC builds upon concepts in the TCSEC but approaches some issues in a different

manner. While differences do exist between the ITSEC and the TCSEC, approximate security classification equivalencies can be made. These differences, however, are sufficient enough to require a product to be re-evaluated with the TCSEC. More details regarding these differences are provided in the ITSEC section.

Like the ITSEC, the Canadian Trusted Computer Product Evaluation Criteria (CTCPEC) also has its beginnings rooted in the TCSEC. Unlike the TCSEC, the CTCPEC's fundamental premise is the separation of functionality and assurance. This has not, however, greatly effected the CTCPEC's intentionally close relationship with the TCSEC. This relationship has enabled many of the evaluations and classifications to be somewhat similar. A direct result of this relationship was the joint venture of updating of the TCSEC into the Federal Criteria

In February 1991, six years after the original introduction of the TCSEC, the Federal Criteria (FC) were released [16.1, 16.12]. The FC builds upon the TCSEC by recognizing and addressing new technology such as operating environments that include multi-window capable workstations [16.1]. These updates were made with the Communications Security Establishment in an attempt to promote international harmonization in the development and use of security classification criteria. Its great success instigated another joint venture which included the authors of the ITSEC and the CTCPEC. This new venture is developing what is known as the Common Criteria.

16.2 The Trusted Computer System Evaluation Criteria

The TCSEC, also known as the *orange book* because of its color, primarily concerns trusted commercially available automated data processing (ADP) systems [16.9]. The criteria classify two evaluation measures, the minimally required security features for a given classification, and the assurance measures associated with a given classification. The security features are primarily intended to evaluated information processing systems that rely upon general purpose operating systems. These features, however, can also be applied to systems in special environments, such as communications processors or process control computers. Unlike the security features, the assurance measures are applicable to all types of computing systems and environments without the need for additional care or attention in their application. Assurance is a means of measuring the confidence that the security features of the system or product accurately enforce and mediate the system or product's security policy—it does what it claims it can do, and it does it well.

The purpose of the TCSEC is threefold [16.9]: (1) to provide a security standard for manufacturers, (2) to provide the DOD with a security metric to evaluate the degree of trust that can be placed on a system, and (3) to provide a basis for product security requirements.

The TCSEC is intended to give manufacturers the necessary information to implement security features in their existing or planned products. This gives manufacturers an idea of how to design a product that meets the security needs of the DOD and other high security organizations before a product is produced. This enables the manufacturer to develop a product that can be considered for wide-spread use in various high security environments. This also reduces any confusion associated with the evaluation criteria and process because the standards are specified in advance of product evaluations.

The criteria also provides the DOD with a security metric for the evaluation process and allows the DOD to fairly and equally evaluate and compare any two or more products. The evaluation process is defined for systems that are environment independent and systems that are environment specific. It should be noted that the evaluation process is not an accreditation. Once a product has been evaluated and given a security classification, it must still be accredited for use in the handling of any classified information.

The TCSEC's third function is to provide a basis for determining the minimal classification required for a given environment. Before a system or product can be selected for use, the environment in which it will function must be examined and classified as well. The environment classification will guide organizations such as the DOD in selecting systems and products that are acceptable. Products that have a classification less than that required by the environment cannot be considered for use.

The classifications defined in the TCSEC are **A**, **B**, **C**, and **D**; the highest, or most comprehensive security division is **A**, the lowest is **D**. Naturally, an unrated system or product is given a **D**. Each division represents a major difference in a product's trust and confidence rating. Within each of these divisions are numbered subdivisions called *classes*. These classes provide a more granular rating in each division. The classification of a product into one of these divisions and classes is based upon four criteria: (1) security policy, (2) accountability, (3) assurance, and (4) documentation [16.9]. While the classification process is quite detailed and relies upon many characteristics, a general description of each division and class is provided in [Table 16.1](#) [16.9].

The TCSEC includes the detailed requirements for meeting each of the security classifications as well as the guidelines for applying the criteria. This includes information concerning the first three security criteria (security policy, accountability, and assurance), a rationale for the division of the classes, supplementary information regarding covert channels, and guidelines for security testing. Further supporting information is provided in the DOD's Rainbow series—a series of documents with multicolored covers that contain detailed explanations and background information for the TCSEC.

Table 16.1. TCSEC security classification criteria descriptions.

Security Classification	General Criteria Description
D	This is the lowest possible classification, and is “catch all” for those products that have been evaluated and failed to meet the requirements of the A through C classifications.
C1	This classification indicates that a product provides need-to-know (discretionary) protection. This is accomplished with a separation of users and data.
C2	Products with this classification provide a more granular access control than those in C1 . This is accomplished with login procedures, audit trails, and resource isolation.
B1	In addition to requiring the features associated with a C2 classification, this classification requires data labeling, mandatory access control over named subjects and objects, and an informal statement of the security policy model.
B2	This protection builds upon B1 by requiring a formal declaration of the security policy, and a more encompassing discretionary and mandatory access control enforcement. Authentication mechanisms must be strengthened for this classification. Covert channels are also addressed in this classification. In general, a B2 system is relatively resistant to unauthorized access.
B3	This classification begins with the B2 classification and requires all user actions be mediated, the system be tamperproof, and the security features be extremely robust and streamline. No additional code or information should be included in the security package. The system must also provide administrator support, auditing, and backup and recovery procedures. In general, a B3 system must be highly resistant to unauthorized access.
A1	A1 classification is functionally equivalent to B3 . The A1 product, however, has gone thorough a more formal analysis that was derived from a formal design and verification of the security features. This analysis must provide a high level of assurance that the system is correctly implemented.

16.3 The Information Technology Security Evaluation Criteria

The ITSEC was created in an attempt to consolidate the many European security evaluation methodologies of information technology products. The ITSEC was a joint security standardization effort that intended to address the needs of both commercial and governmental security products. The current issue, version 1.2, was published in 1991 and is quite detailed. This discussion is provided to give the reader a high level understanding of the ITSEC. Readers interested in learning more about the ITSEC are encouraged to visit the ITSEC home page on the Internet at <http://www.itsec.gov.uk/>.

The ITSEC standardization effort is intended to be a super-set of the TCSEC, with ratings translatable to those of the TCSEC [16.8]. To achieve the super-set status, the ITSEC separates the concepts of security functionality and assurance assessment. Each product is given at least two ratings, at least one that indicates the implemented security functionality and one that indicates the correctness of the implementation.

The functionality criteria are measured on a rating scale from **F1** to **F10**, with **F1** being the lowest rating [16.3, 16.8]. An ITSEC rating of **F1** roughly corresponds to a TCSEC rating of **C1**, with each successive rating of the ITSEC corresponding to the successive ratings in the TCSEC. The **F6** through **F10** functionality ratings add the following concepts: data and program integrity (**F6**), system availability (**F7**), data communication integrity (**F8**), data communication confidentiality (**F9**), and network security including confidentiality and integrity (**F10**) [16.8]. A product will receive a base rating from **F1** to **F5** plus any combinations of ratings from **F6** through **F10**. For example, a product may be functionally rated at **F3**, **F7**, and **F8**. The detailed evaluation process can be found in the formal ITSEC standards; they are too large and complex to present here.

The correctness criteria are used to evaluate a product's level of assurance. An assurance rating corresponds to the effectiveness and added value to the security process. The ratings, from lowest to highest, are: testing (**E1**), configuration control and controlled distribution (**E2**), access to detailed design and source code (**E3**), extensive vulnerability analysis (**E4**), demonstrable correspondence between design and source code (**E5**), and formal models and descriptions with formal correspondences between the two (**E6**). These assurance ratings are cumulative. That is, a product with an **E3** correctness evaluation has also passed **E1** and **E2** evaluation.

The two ratings, functionality and assurance, compose an ITSEC security classification that can be coarsely translated to a TCSEC classification. This translation is provided in [Table 16.2](#). Because the TCSEC classifications do not provide any direct correspondence to the ITSEC's **F6-F10** classifications, they are not included in the translation table. It should also be noted that these translations are not bi-directional. Because of the additional subdivisions in security classifications

resulting from the separation of functionality and assurance, a TCSEC security classification does not uniquely map to an ITSEC classification.

Table 16.2. Translating ITSEC classifications to TCSEC classifications.

ITSEC Classifications		TCSEC
Functional	Assurance	Classifications
	E0	D
F1	E2	C1
F2	E2	C2
F3	E3	B1
F4	E4	B2
F5	E5	B3
F5	E6	A1

The existence of the mappings in [Table 16.2](#) does not mean that the TCSEC evaluation process can be bypassed. It is simply a means of approximating the associated classifications. The approximated translations, however, should be easily attained through the proper TCSEC evaluation process.

16.4 The Canadian Trusted Computer Product Evaluation Criteria

In May 1989, the Canadian System Security Centre (now known as the Communications Security Establishment) released the CTCPEC in response to the Canadian Treasury Board's Government Security Policy [16.4]. Like the other security evaluation metrics, the CTCPEC evaluates the effectiveness of a product's security. This criteria was designed for governmental use without the intention of providing a direction of growth for commercial products. Because the criteria provide information concerning the Canadian government's evaluation process, however, it is still of value to a developer.

The CTCPEC, like the ITSEC, divides the security requirements into two groups: functional requirements and assurance requirements [16.4]. The functionality requirements are further divided into four policy categories: (1) confidentiality, (2) integrity, (3) availability, and (4) accountability. Each category consists of security requirements that should prevent specific system threats. Each security requirement is

further divided into a pre-defined set of rating levels that indicate a product's effective security. These ratings range from zero to five—in most cases, only four—where zero indicates a failure to meet a requirement. These functional requirements, and the associate rating levels, are listed in [Table 16.3](#).

The functional security requirements listed in [Table 16.3](#) are not necessarily independent of each other [16.4, 16.5]. That is, some requirement ratings are dependent upon other requirement ratings. This was intentionally planned and included in the design of the CTCPEC as the proper implementation of many security features requires the presence of other security features. For example, the CTCPEC states that a product that successfully meets the requirements of a **CD-1** rating, must have first met the **WI-1** requirements. Additional constraints are made more apparent in the CTCPEC document.

Table 16.3. The CTCPEC functional requirements and their rating ranges.

Accountability	Range
Audit	WA-0 to WA-5
Identification and Authentication	WI-0 to WI-3
Trusted Path	WT-0 to WT-3

Availability	Range
Containment	AC-0 to AC-3
Fault Tolerance	AF-0 to AF-2
Robustness	AR-0 to AR-3
Recovery	AY-0 to AY-3

Confidentiality	Range
Covert Channels	CC-0 to CC-3
Discretionary Confidentiality	CD-0 to CD-4
Mandatory Confidentiality	CM-0 to CM-4
Object Reuse	CR-0 to CR-1

Integrity	Range
Domain Integrity	IB-0 to IB-2
Discretionary Integrity	ID-0 to ID-4
Mandatory Integrity	IM-0 to IM-4
Physical Integrity	IP-0 to IP-4
Rollback	IR-0 to IR-2
Separation of Duties	IS-0 to IS-3
Self Testing	IT-0 to IT-3

The second part of the evaluation process is the evaluation of a product's assurance requirements. A product will receive an assurance rating, from a low of **T-0** to a high of **T-7**, based upon meeting the assurance requirements dictated in the

CTCPEC. The assurance requirements concern the following: architecture requirements, development environment requirements, development evidence requirements, operational environment requirements, documentation requirements, and testing requirements [16.4]. These requirements dictate how a product is to be designed, how a product is to be prepared for evaluation, how a product is to be provided to a customer, and how a developer is to indicate that a product's trust has been sufficiently tested. The assurance rating resulting from this evaluation, in addition to a functional rating, completes a product's security rating.

Like the ITSEC ratings, the CTCPEC ratings can be translated to equivalent TCSEC ratings. Approximate TCSEC equivalencies are indicated in [Table 16.4](#) [16.4, 16.7]. Because of the more granular ratings of the CTCPEC, this translation is neither exact nor bi-directional.

Table 16.4. Translating a CTCPEC rating onto a TCSEC rating.

CTCPEC Rating Group	Approximate TCSEC Rating
WA-1, WI-1, CD-2, CR-1, ID-1, IS-1, IT-1	C2
WA-1, WI-1, CD-2, CM-2, CR-1, ID-1 or IM-1, IS-1, IT-1	B1
WA-1, WI-1, WT-1, CC-1, CD-2, CM-3, CR-1, ID-1 or IM-1, IS-2, IT-1	B2
WA-2, WI-1, WT-2, AY-1, CC-1, CD-3, CM-3, CR-1, ID-1 or IM-1, IS-2, IT-1	B3

16.5 The Federal Criteria

The Federal Criteria for Information Technology (FC) was began in an attempt to update the TCSEC [16.10]. The original goal of the Federal Criteria was to create a national security standard that protects the existing investment in security technology, improves the existing security evaluation process, plans for the changing needs of the customer, and promotes international harmonization in security evaluation. While the

FC never went beyond the “draft” stage, it is generally considered to be a successful improvement to the TCSEC [16.10, 16.11]. Even after several modifications since its initial release in December 1992, the FC has never been formally completed or adopted as a replacement to the TCSEC. Its creation, however, did initiate efforts to create the Common Criteria.

The FC addresses its goals with the introduction of the protection profile. A protection profile is a set of criteria that define a specific level of security and trust for a given product. This allows new profiles to be created as technology progresses and the security community’s needs change. Profiles are stored in a central location, a registry, so that developers, evaluators, and consumers can easily obtain new profiles that are guaranteed to be accurate and up to date.

A profile consists of a functional component, a development assurance component, and an evaluation component. The functional component identifies the features that a product must support to meet the profile. Possible functional components include features like auditing, access controls, and trusted paths. As the design of the FC is highly dependent upon both the TCSEC and the CTCPEC, the functionality discussed in each of these criteria were included in the various FC profiles. The development assurance components dictate the degree to which a product must support its design, control, and use. This is similar to the assurance controls used in the CTCPEC. The evaluation assurance components of a profile are the assurance measures a product undergoes to verify the trust and security it claims to provide. Some evaluation assurance components include such items as security testing and covert channel analysis. This separation of evaluation assurance and development assurance is similar to the T-level provided by the CTCPEC, but the inclusion of evaluator assurance is more closely related to the ITSEC. Unfortunately, the increased number of possible assurance combinations resulted in numerous similar profiles. This has resulted in overly complicating the evaluation and rating process.

While the rating process is difficult, it is possible to create an approximate translation of the FC’s assurance ratings to those of the TCSEC and the CTCPEC. Not surprisingly, they are quite similar to the T-levels of the CTCPEC. [Table 16.5](#), below, shows this relationship.

Table 16.5. Approximate assurance rating translations of the TCSEC, FC, CTCPEC, and the ITSEC.

TCSEC	FC	CTCPEC	ITSEC
D			E0
C1			E1
C2	T-1	T-1	E2
B1	T-2	T-2	E3
	T-3	T-3	
	T-4		
B2	T-5	T-4	E4
B3	T-6	T-5	E5
A1	T-7	T-6	E6
		T-7	

The attempt to force commonalities between the trust evaluation process of the TCSEC and the CTCPEC produced what many consider to be an improved, but flawed criteria. The greatest benefit of this collaboration was not the resulting Federal Criteria, rather the decision to begin the collaboration process anew with input from the creators of the ITSEC. This collaboration resulted in the creation of the Common Criteria.

16.6 The Common Criteria

The Common Criteria (CC) is an attempt to combine the many existing security criteria into a unified standard [16.6]. This process began in 1993 and the first draft was completed by the Common Criteria Editorial Board (CCEB) in January, 1996. Version 1.0 was put through many trial tests and review processes. The comments resulting from its trial use became the basis for a significant revision by the newly created Common Criteria Implementation Board (CCIB), which replaced the CCEB. In October, 1997, version 2.0 Beta of the CC was released for comment. The result of this processes is the now created, and widely available, CC version 2.0, formally called the “Evaluation Criteria for Information Technology Security.” It should be noted that this is by no means the final version of the CC; it is designed to grow and evolve as technology and other requirements change. The remainder of this discussion, however, will focus on the current version of the CC. This discussion is provided to give the reader a high level understanding of the CC—version 2.0 is over 300 pages in length.

Readers interested in learning more about the CC are encouraged to visit the CC Project's home page on the Internet at <http://csrc.nist.gov/cc/>.

The current version of the CC is designed to be beneficial to the consumers, designers, and evaluators of the product being reviewed, or the **Target of Evaluation** (TOE). Consumers can use the evaluation results to determine if a TOE meets their requirements. These requirements are typically defined in implementation independent descriptions called **Protection Profiles** (PP). The developer can use the CC evaluations to prepare for a product evaluation. By identifying the desired security features, a product can be modified during its development. The security requirements and specification information that a product is tested against is called a **Security Target** (ST). The ST is the basis that each TOE is evaluated against. The CC aids an evaluator in describing the actions that are to be undertaken to evaluate a product against the various STs. In all, the CC provides a very clear evaluation process for all security related products.

The CC is divided into three parts: (1) the model description, (2) the security functional requirements, and (3) the security assurance requirements. The model description defines the general concepts of the CC, the evaluation process, and the constructs for properly identifying the requirements and specification of a TOE. The functional requirements section establishes a set of functional components for describing a TOE. The assurance requirements define a set of components for evaluating the assurance of a TOE. This includes a predefined scale of Evaluation Assurance Levels (EALs).

According to the model, for a TOE to be evaluated, it must be provided to the evaluation committee along with the desired ST, and evidence that the TOE meets the ST requirements. The results of this process are evaluator reports concerning the findings and, hopefully, confirmation that the TOE meets the desired ST level. If the TOE is confirmed to meet the desired ST, an EAL will also result from the evaluation process.

The CC builds upon the functional ratings found in other criteria by combining many of the concepts of the existing criteria and extrapolating the rest. The result is a set of functional requirements that have a hierarchical relationship. The highest level in the hierarchy is composed of objects called **classes**. There are eleven classes in the CC (see [Table 16.6](#)), each of which has a security-related focus, such as audit, cryptography, or privacy. The class contains information regarding its purpose and use. This information is provided to assist a user's understanding and application of the CC. Each class is further subdivided into member topics that relate to the functional operation of the class.

Table 16.6. The functional classes of the Common Criteria.

Functional Class Name (Selected Abbreviation)	Number of Family Members
Security Audit (FAU)	6
Communication (FCO)	2
Cryptographic Support (FCS)	2
User Data Protection (FDP)	13
Identification and Authentication (FIA)	6
Security Management (FMT)	6
Privacy (FPR)	4
Protection of the TOE Security Functions (FPT)	16
Resource Utilization (FRU)	3
TOE Access (FTA)	6
Trusted Path/Channels (FTP)	2

The members, or *families*, of any given class are all related to the classes security focus. For example, the families in the CC's 'Security Audit' class include 'Audit Automatic Response', 'Audit Analysis', and 'Audit Selection.' The eleven CC classes contain a total of 76 families. Each family can be further divided into *components*.

Components are the smallest selectable set of specific security requirements relating to a given family. The number of components in a family may vary; in some cases there is only one. An example of a component from the Security Audit Review Family is 'the capability to read information from the audit records.' Each component contains information regarding how its presence and functionality in the TOE can be audited (or verified), how the component should be managed, and if there are any relationships to other components. Some components depend on the presence of other components, for example, the "Audit Review" component depending on the presence of the 'Audit Generation' component.

The assurance requirements of the CC are presented in a hierarchical structure similar to that of the functional requirement; multiple classes, each containing multiple families, with at least one component. There are seven assurance classes, each of which are further divided into a total of 26 assurance requirement families. [Table 16.7](#) lists the assurance requirement classes of the CC [16.6]. Like functional requirements, the number of assurance requirements depends on what the criteria's creators identify as necessary. The design of the assurance classes and families also mimics the

functional classes with the inclusion of supporting information. This includes component objectives, user notes, and user responsibilities.

Table 16.7. The assurance classes of the Common Criteria.

Assurance Class Name	Number of Family Members
Configuration Management (ACM)	3
Delivery and Operation (ADO)	2
Development (ADV)	7
Guidance Documents (AGD)	2
Life Cycle Support (ALC)	4
Tests (ATE)	4
Vulnerability Measures (AVA)	4

A TOE's assurance evaluation consists of up to 26 individual ratings; one for each of the 26 assurance families. The CC identifies seven assurance level ratings, EAL1 through EAL7; EAL1 representing lowest amount of assurance. A description of the EALs is provided in [Table 16.8](#). For a TOE's member family to receive a specific rating, it must meet the assurance requirements for the given level. To increase the assurance from one EAL to another EAL the TOE must be modified either substituting a component from within the same family or augmenting the TOE with components from other families.

Table 16.8. Evaluation Assurance Level (EAL) descriptions.

Evaluation Assurance Level	Description
EAL1	Functionally Tested
EAL2	Structurally Tested
EAL3	Methodically Tested and Checked
EAL4	Methodically Designed, Tested, and Reviewed
EAL5	Semi-Formally Designed and Tested
EAL6	Semi-Formally Verified Design and Tested
EAL7	Formally Verified Design and Tested

Because of the structure of the CC, not every assurance rating is applicable to every family. For example, it does not make sense to give a ‘Functionally Tested’ rating for the ‘life cycle definition’ (ALC_LCD) of a TOE. As such, ALC_LCD cannot receive a rating of EAL1–nor can it receive an EAL2 or EAL3. The CC provides a detailed list of the possible ratings each assurance family can receive.

Before a TOE is evaluated, however, the PPs and STs are typically evaluated. PP and ST evaluation occurs first so that they can be verified to be a meaningful basis for evaluation and comparison of the TOEs. The goal of the PP and ST evaluation is to demonstrate that they are complete, consistent, and sound, and thus a reasonable statement of requirements for a TOE. The evaluation of PPs and STs is almost identical to the process presented above for TOEs. The evaluation process, however, has special assurance classes that it must follow. The classes force the author to formally detail the PP by describing the security environment, security objectives, and the IT security requirements, amongst other things. This results in a clearly described and unambiguously defined PP and ST.

Version 1.0 of the CC contained example PPs that provided a means of translating security evaluations between the various security criteria and the CC. These translations have not yet been carried forward to version 2.0 of the CC. Until a formal translation between the CC and other security criteria is released, it is recommended that persons responsible for performing security evaluations start anew with the CC.

16.7 British Standard 7799

In 1995 the British Standards Institution (BSI) recognized the need for a common security framework that would enable companies to develop, implement, and measure effective security management practices. Using the existing code of practice for information security management, PD0003, the BSI developed British Standard 7799 (BS7799). This standard is based on the best information security practices of leading British and international organizations. BS7799 provides a comprehensive set of security controls that is intended to serve as a single reference for security issues that should be addressed by governmental organizations and businesses alike.

The standard is an excellent beginning for the creation of a secure environment. It does not address every security issue, nor does it provide an inexperienced security administrator everything necessary to achieve the robust security environment that is so often desired. It is merely a set of recommendations and a road map to success. As the standard itself states, “[BS7799] is intended for use as a reference document for managers and employees who are responsible for initiating, implementing and maintaining information security within their organization” [16.2].

The BS7799 security model identifies a set of three primary security issues (*confidentiality, integrity, and availability*) and then presents a list of high-level security objectives that address these issues. BS779 is divided into ten functional sections, each detailing a specific high-level security objective. Each section contains a comprehensive list of security controls that support the objective of the section. The ten sections, and their objectives are listed in [Table 16.8](#) [16.2].

Table 16.8 BS7799 Security Objectives.

Section	Security Objective
1	Security Policy
2	Security Organization
3	Assets Classification and Control
4	Personnel Security
5	Physical and Environmental Security
6	Computer and Network Management
7	System Access Control
8	System Development and Maintenance
9	Business Continuity Planning
10	Compliance

The controls within each section comprise the supporting forums, documentation, or activities that, together, achieve the objective of the section. It should be noted that some controls are not applicable to every environment or situation and should therefore be used selectively. It is recommended, however, that all applicable controls be met as they define what BS7799 calls “an industry baseline of good security practice.” That is, they are the equivalent of what many refer to as “Industry Best Practices.” While there are 109 security controls, ten of them are further identified by BS7799 as key controls. These key controls are considered to be very important and necessary to develop an information security management plan, but no means form a complete plan. The ten controls are listed and described in [Table 16.9](#) [16.2].

Table 16.9. The ten key controls of BS7799.

Control Number	Security Objective	Brief Description
1.1.1	Information Security Policy Document	All employees responsible for any portion of information security should have a copy of the information security policy.
2.1.3	Allocation of Information Security Responsibilities	The responsibility for protecting specific assets and for implementing security processes to protect those assets should be clearly defined within an organization.
4.2.1	Information Security Education and Training	All system users should be given security awareness education and the appropriate technical training.
4.3.1	Reporting of Security Incidents	Security incidents should be reported to predefined management representatives as soon as possible.
6.3.1	Virus Controls	Virus detection and prevention mechanisms should be used and user awareness programs implemented.
9.1.1	Business Continuity Planning Process	A process should exist for the creation, maintenance, and testing of the organization's business continuity plan.
10.1.1	Control of Proprietary Software Copying	Implement and enforce a strict policy against software piracy which includes user awareness education.
10.1.2	Safeguarding of Organizational Records	Sensitive and critical organization documents should be protected from loss, destruction, and falsification.
10.1.3	Data Protection	Applications handling personal information should meet any and all legislation.
10.2.1	Compliance with Security Policy	Policy and standards compliance testing should occur in every area of the organization.

BS7799, like many of the other security standards, is constantly undergoing change due to technology and necessity. In 1998, a second part of the standard was created; it describes the requirements for establishing, implementing, and documenting

systems as modeled in the original standard. Part 2 of BS7799 is primarily used as a requirements specification for organizational assessment, compliance, and certification. Part 2 also forms the basis for a security assessment of an organization's information systems. It is often used as the basis for formal certifications such as "c:cure," a new BS7799 accredited certification scheme.

The latest update to BS7799 is PD0007, "the Guide to the British Standard Code of Practice for Information Security Management." It is a direct update to the concepts of PD0003, and thus an update of the first part of BS7799. Naturally, as modifications are made to this, or any other part of BS7799, the accreditation process will also change. This standard and certification process are designed to be easily updated

16.8 Summary

Security standards such as the TCSEC were created with the intent of ensuring that trusted products achieve a measurably high degree of security. This has become increasingly necessary as organizations, such as a government, require products that can be trusted to maintain a secure environment and confidential information. In response to this need, the CTCPEC, the ITSEC, and the FC were also created by geographically different organizations. The various standards identify a definition of acceptable levels of security and trusted computer systems. The difficulty with evaluating a product with any one of these established security standards is that a potential customer may not share the same set of standards as a developer, not to mention the product evaluator. In an attempt to reduce this possibility, the international security community has begun to develop the Common Criteria; a set of security evaluation standards that define a globally accepted level of trust and integrity. The long term success of the Common Criteria will not only depend on how it is accepted by developers, evaluators, and authors, but on its ability to adapt to an ever maturing technology. This has not shown itself to be a problem as the Common Criteria has undergone significant changes since its inception.

16.9 Projects

- 16.1 How does reducing the number of security standards effect a developer's burden in creating new trusted security products?
- 16.2 What are the negative impacts of combining the many existing trusted computer evaluation criteria into a single criteria.

- 16.3 In what respects do each of the CTCPEC, TCSEC, and ITSEC provide a more complete form of trusted product evaluation?

16.10 References

- 16.1 Basic, Eugen Mate, "The Canadian Criteria, Version 3.0 & the U.S. Federal Criteria, Version 1.0", *Proceedings of the Fifth Annual Canadian Computer Security Symposium*, Ottawa, Canada, 1993, pp. 537-548.
- 16.2 British Standards Institution, Code of Practice for Information Security Management, BS 7799:1995.
- 16.3 Brouwer, A; Casey P.; Herson, D.; Pacault, J.; Taal, F. and Van Essen, U., "Harmonized Criteria for the Security Evaluation of IT Systems and Product", *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 394-403.
- 16.4 Canadian System Security Centre, The Canadian Trusted Computer Product Evaluation Criteria, Version 3.0e, January 1993.
- 16.5 Cohen, Aaron and Britton, Kris, "Comparison of CTCPEC and TCSEC Rated Products", *Proceedings of the Sixth Annual Canadian Computer Security Symposium*, Ottawa, Canada, 1994, pp. 457-469.
- 16.6 Common Criteria Implementation Board, Common Criteria for Information Technology Security Evaluation, Version 2.0, October 1997.
- 16.7 Gibson, Virgil and Fowler, Joan, "Evolving Criteria for Evaluation: The Challenge for the International Integrator of The 90s", *Proceedings of the 15th National Computer Security Conference*, Baltimore, Maryland, October 1992, pp. 144-152.
- 16.8 Information Technology Security Evaluation Criteria, Harmonized Criteria of France, Germany, the Netherlands, and the United Kingdom, Version 1.2, May 1991.

- 16.9 National Computer Security Center, Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD, December 1985.
- 16.10 National Institute of Standards and Technology and the National Security Agency, Federal Criteria for Information Technology Security, Vol. 1 and 2, December 1992.
- 16.11 Neumann, Peter G., “Rainbows and Arrows: How the Security Criteria Address Computer Misuse”, *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 414-422.
- 16.12 Schwartau, Winn, “Orange Book II: The New Federal Criteria”, *InfoSecurity News*, MTS Training Institute Press, Framingham, Massachusetts, Vol. 4, No. 4, July/August 1993, pg. 74.

16.11 Extended Bibliography

- 16.13 Branstad, Martha A.; Brewer, David; Jahl, Christian; Pfleege, Charles P. and Kurth, Helmut, “Apparent Differences between the U.S. TCSEC and the European ITSEC”, *Proceedings of the 14th National Computer Security Conference*, Washington, DC, October 1991, pp. 45-58.
- 16.14 Cohen, Aaron, “Modeling the Government of Canada Security Policy”, *Communications Security Establishment Technical Report*, Ottawa, Canada, May 1994.
- 16.15 Common Criteria Evaluation Board-94/080, Common Criteria for Information Technology Security Evaluation, Version .9, October 1994.
- 16.16 Straw, Julian, “The Draft Federal Criteria and the ITSEC: Progress Toward Alignment”, *Proceedings of the 16th National Computer Security Conference*, Baltimore, Maryland, October 1993, pp. 311-323.

17

CASE STUDIES

To better illustrate the need for security three specific incidents are presented followed by a partial list of known and unknown hacker incident over the past 20 years. Each of the incidents addresses a different aspect of computer security. The first incident, the Hanover Hackers, proves two important facts: (1) intrusive activity has grown to global proportions, and (2) system vulnerability often results from poorly trained or apathetic system administrators. The second incident concerns a virtual jail that was created for the purposes of restricting and observing an intruder's actions. This was done to gain a better understanding of the tricks and techniques used by intruders. The third incident, the Internet Worm, provides insight into the operations of a worm. This incident also shows that sophisticated techniques are not necessary for executing wide spread attacks.

It should be noted that this discussion is not intended to be a how-to guide. It is provided with the intent that the knowledge gained by learning and understanding prior, successful attacks will be useful in preventing future attacks.

17.1 The Hanover Hackers

It started with a 75 cent accounting discrepancy¹. Clifford Stoll was tasked with tracking down a computer accounting error at the Lawrence Berkeley Laboratory (LBL) computer facility—an unclassified research center at the University of California, Berkeley. The discrepancy was tracked to a new computer account that did not have the required billing information. LBL's initial belief that this was the work of an intruder was confirmed when the National Computer Security Center informed LBL that someone from their lab had attempted to break into their network. Even though

¹ The following account is based on Clifford Stoll's recount of the incident in [17.12].

LBL removed the offending account, continued suspicious activity made it clear that the intruder was still in the system.

Unbeknownst to Stoll, LBL, and other victims, the hacker had been using simple techniques to obtain access to the various systems and networks. His methods focused on guessing passwords and taking advantage of now patched vulnerabilities in both the *sendmail*² and *emacs*³ programs. Password guessing was most successful at gaining access to default accounts that were shipped with the system and never changed. These accounts were then used to gain system administrator, or root⁴, capabilities by exploiting the *sendmail* and *emacs* vulnerabilities.

Once inside a system, the intruder would hide by appearing as an authorized user or administrator. The intruder would then search for interesting data and attack other systems by examining files for the presence of specific words or phrases. The process starts with the disabling of audit logs that record user activity. The hacker would then plant a Trojan horse program to capture passwords. The hacker's next course of action would be to attack other systems using the same simple techniques. Successful attacks were made on at least eight US military sites while the intruder had access to the LBL system. A more detailed picture of the intruder's means of access is provided in [Figure 17.1](#).

In an attempt to catch the intruder and learn more about the intruder's techniques and interests, LBL decided to track and observe him. Whenever the intruder logged into the system, LBL began to trace the phone line. Because the advanced phone services that are readily available today did not exist, special modifications to the system were required to detect the intruder and recorded all activity. Line printers and personal pagers were also connected to the system. The line printers recorded all network communications with the intruder and modems dialed the pagers when the intruder logged on. These tools became very important to the success of the tracking. Unlike software, the hardware could neither be modified nor detected by the intruder. The operators would therefore be immediately informed of the intruders presence at any hour of the day and the intruder could not prevent it. Additionally, the non-modifiable activity logs recorded on the printers would help to record the activity of the intruder and possibly provide insight to his thinking and approach. For these benefits

2 The *sendmail* program implements the SMTP protocol (see footnote 9). The DEBUG problems that exist in the SMTP protocol exist in the *sendmail* program as well. This security hole allows a user to execute commands on a remote computer system.

3 The Gnu *emacs* editor is a widely used versatile text editing program. Its versatile design includes, among other things, its own mail system. The mail capability is shipped with portions operating with superuser privileges. This can be used to change file ownership and move files into reserved system areas as well as other user's directories.

4Root, or superuser, is the system administrator's account. This account has complete control over all aspects of the computer system [17.6].

to be realized to their full potential, however, the intruder would have to keep coming back. As long as LBL provided interesting bait, this was not a concern.

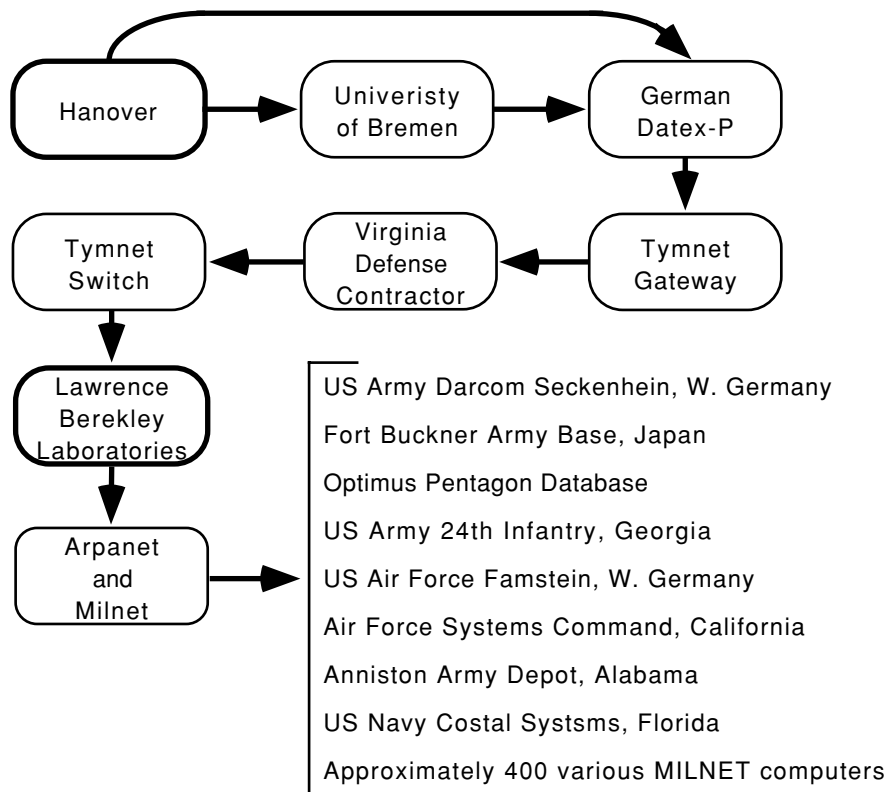


Figure 17.1. The Hanover Hacker's system access route and destinations.

As expected, the intruder returned many times. Unfortunately, he only remained connected for short periods of time—approximately two minutes. Nonetheless, the intruder was traced to a connection in Oakland, California. The intruder was traced from Oakland to a dial-up modem at a Virginia defense contractor. Close inspection of the contractor's phone records indicated that the intruder had been accessing the contractor's system months before he was detected at LBL. The contractor responded by restricting access to their system. While this did stop the intruder from accessing the contractor's system, he was still able to use other access routes to LBL and other various targets. While this did slow down the process, the intruder was still traced back to Hanover, Germany.

While the tracing enabled Stoll and the authorities to determine the city of origin, unsophisticated technology accompanied by short connection times prevented the telephone companies from getting more precise in their tracking. The German telephone company required a long connection time so that they could manually check their older mechanical switches⁵ that route phone calls. To achieve this, Stoll's group created volumes of fictitious data that would appeal to the hacker. Their hope was that the information would seem valuable enough to warrant longer connections. The data, therefore, contained key words that suggested that the information was about the US Government's Strategic Defense Initiative (SDI). As hoped, the volumes of fake memos, personal e-mail, mailing lists, contact points, and general research was too much for the hacker to resist. The hacker spent enough time on a single connection so that a successful trace back to the hacker's home was possible. Shortly thereafter, the hacker and four others were arrested on charges related to the intrusions.

The German Government charged the five men with espionage—specifically, selling information to the Soviet KGB. The primary hacker and two others were each to receive a maximum of two years in prison and fines up to \$12,000. One of the remaining men cooperated with the state in exchange for immunity. The fifth man was found dead of suspicious causes before the trial began.

17.2 An Evening With Berferd

With the history of phone phreaks⁶ attacking the American Telephone and Telegraph Company (AT&T), it should come as little surprise that AT&T Bell Laboratories takes both computer and phone security seriously. In an attempt to reduce the number of incidents into their network, they installed a secure Internet gateway and reduced the number of computer services supported by their system [17.2]. This still did not stop unauthorized attempts to access their systems and network. In one such incident, a member of AT&T's technical staff, Bill Cheswick, decided to determine who was trying to break in, where they were coming from, and how they were trying to compromise the existing security.

To help satisfy their curiosity, Bell Labs added some fake services to their system and wrote the necessary programs to scan the resulting log files. The additional

⁵ Before small computers were implemented to automatically route telephone calls, large mechanical switches were used. Since there were no electronic means of determining the routing established by a telephone call connection, a person would have to manually locate the proper bank of switches and decipher the caller's phone number.

⁶ *Phone Phreaks* are people with more than a passing interest in the operations of the phone system. They have knowledge and capability to control every aspect of the telephone system. Their knowledge includes the ability to alter billing records, alter a customer's services, reroute telephone calls, and establish or disconnect service as they so desire [17.7].

services included the File Transfer Protocol⁷ (FTP), *telnet* access⁸, guest user accounts, and SMTP DEBUG⁹. The FTP services were installed such that attempts to obtain the FTP password files were reported. In addition, the service would report attempts to exploit older FTP bugs. Like the FTP services, all *telnet* and login attempts, either failed or successful, were recorded and reviewed. Since the gateway provided limited access, locating unauthorized accesses would be simple. The *rsh*¹⁰ and *rlogin*¹¹ services were completely disabled as they rely on a poor authentication system. Attempted use of these services, however, was reported. In a similar manner, the *finger*¹² services were disabled. Guest accounts were modified and installed such that they reported their activation and made the system appear to be very busy. The final modification was the introduction and alteration of an old *sendmail* vulnerability that allowed outsiders to execute a shell script as root¹³. In the event that the *sendmail* DEBUG vulnerability was exploited, the intruder's command would be reported to the AT&T group. In addition, each of these services attempted to locate the attacker by performing a reverse finger to the attacker's computer. All that was left to do was wait for something to happen.

While many probes of the system did occur, it was not until seven months after the security setup had been installed that something of significance happen. An intruder originating from Stanford University located the *sendmail* vulnerability and requested the system to mail back a copy of its password file. Having a bogus password file readily available, Cheswick mailed it to the intruder. The next mail

7 The *File Transfer Protocol* (FTP) is a method of copying files from one computer system to another. FTP requires the user to have an account on each of the systems. Some systems provide 'anonymous', or 'guest', accounts that allow anyone to access the system. The privileges given to anonymous accounts are normally restricted as there is limited control over their use [17.11].

8 *Telnet* allows a user to log on to a computer system from across a computer network. Telnet may operate between two computers running different operating systems [17.11].

9 The *Simple Mail Transfer Protocol* (SMTP) is the means of communication used by computer systems to send mail back and forth. One of SMTP's few commands, DEBUG, puts the protocol into a debugging mode. This mode has the ability to allow a user to specify commands for the system to execute as opposed to performing mail related actions [17.11].

10 The *rsh* (or *rmsh*) command executes a command on a remote computer system. The remote system executes the command as if it were requested locally and redirects any output back to the user. As with *rlogin*, *rsh* only executes on trusted systems [17.1, 17.3].

11 *Rlogin* was initially intended for remote logins between two UNIX systems. Since its inception, however, it has been ported to include several non-UNIX systems. *Rlogin* requires the local user and system to be trusted by the remote system. That is, the remote system must have previously granted a user permission to access the system from their local computer. Since permission must be explicitly set, a user is often not prompted for a user name or password [17.6].

12 The *finger* service allows a user to obtain information about other users. This information often includes the user's real name, their home director, their last login time, and when the user last read and received mail. This service is under scrutiny for two reasons. First, its use may be an invasion of privacy as a user has limited control over the information returned by a *finger* command. Second, a bug in the service played a significant role in the deployment of the Internet Worm (see chapter 17.3) [17.3].

13 *Root* -See 4.

Cheswick sent was to both Stanford and the Computer Emergency Response Team¹⁴ (CERT), informing them of the intruder's activities. Stanford replied that they were already aware of the problem and that the offending user account had been stolen. A few days later, Cheswick received mail that the bogus password file had shown up in a user account in France. The intruder returned that same day.

The intruder's presence was signaled when he attempted to finger an account named 'berferd.' A few minutes later, the intruder attempted to make a new account, 'berferdd.' Deciding that it would be worth the time to mislead the intruder and make the action appear successful, Cheswick created the account to see what would happen next. This required Cheswick to manually respond to many of the intruder's resulting actions since most of the system services were disconnected or falsified.

For Cheswick to be able to masquerade as the system and maintain a level of consistency in his responses, he had to make a few decisions about the state and operation of the system. The first being that the bogus FTP password file had to be real. This led to the second decision; the password file and *sendmail* hole would exist because general administration of this system was going to be considered poor. The third decision was to be that the system was excruciatingly slow. Realizing that he could not manually respond to the attacker as fast as system services, Cheswick felt it was the most plausible reason for any delay.

Cheswick's delay in creating the 'berferdd' account prompted the intruder (referred to by Cheswick and thus herein as 'Berferd') to attempt to create another account, 'bferd.' While Cheswick installed the new account, Berferd's impatience got the best of him and he attempted to use the *talk*¹⁵ command. Not wanting to support *talk*, Cheswick's next decided was that the system did not have the command. This led Berferd to create another account, 'bferd,' and attempt to *rlogin* into the system. After failing to *talk* and *rlogin*, Berferd requested a copy of the *inetd.conf*¹⁶ file from the system. Cheswick, realizing the importance of this file, did not want to give it up nor did he want to spend the time creating a fake one. This led to the decision that the system was non-deterministic. Thus, the file was never sent to Berferd. Following Berferd's next command, a failed attempt to see list all of the tasks the computer was performing, Cheswick decided to get CERT more involved. CERT placed

14 The *Computer Emergency Response Team* (CERT) was established to respond to Internet based intrusions and other security concerns. CERT attempts to analyze security vulnerabilities and report and distribute the necessary security patches [17.1].

15 *Talk* allows two users to interactively communicate with each other in real-time. Each users terminal is divided into an incoming and outgoing region. All incoming information from the remote user is displayed in one region, while everything a user types is sent and displayed in the outgoing region. The end result is the equivalent of a typed telephone call [17.6].

16 The *inetd*, or Internet services daemon, is responsible for overseeing most network operations. This is accomplished by managing several specialized services such as FTP, *rlogin*, and *telnet*. These services are listed in the *inetd* configuration file, *inetd.conf*. This file indicates the manner in which each of the services operates and how they are activated and possibly protected [17.5].

Cheswick in contact with the system administrators at Stanford and also began to log and monitor everything. After another hour of watching Berferd poke around the system and modify the finger program, Cheswick decided to call it a night. He sent out warnings of possible disk errors and shut down the system. Approximately half an hour later, Cheswick brought the machine back on line so that mail could get through.

As expected, Berferd returned later that evening. Review of the log files revealed that Berferd decided to cover his tracks and execute the necessary commands to erase all of the system files. The next morning, Berferd returned and again tried to erase the system files. In an attempt to simulate a successful erasure of the files, Cheswick took the machine down, cleaned up the modified files and posted a general login message that there was a disk crash. Berferd returned that afternoon, and a few more times over the following week. Cheswick's response time had dropped significantly, especially since he was often unable to respond more than once or twice a day. Berferd's persistence and patience clearly meant that he believed he had found a valuable system to penetrate.

With the help of others at AT&T Bell Laboratories, Cheswick set up a software version of a jail for Berferd. This jail would allow them to monitor Berferd's activities, learn Berferd's techniques, and warn any victims without the risk of losing valuable data. In an effort to hide the effects of being in a jail, they removed any software that could potentially reveal the presence of the jail. They also prepared a believable, but fake, login script and file system that contained tempting files. Hopefully, Berferd would be interested enough in the files to spend enough time for a line trace to be made.

As hoped, Berferd spent considerable time in the jail. Once inside, he was tracked to a Stanford modem line. The next step was to obtain a phone trace. The trace indicated that the calls were coming from the Netherlands. Further tracing of Berferd was impossible, however, as Dutch law did not make computer hacking illegal.

Berferd's presence in the Bell Lab's jail had become less frequent. At the request of the management, the jail was eventually shut down. Berferd failed to gain access to the system a few more times before he stopped trying.

While prosecution was not the end result of this activity, the jail did serve to provide a reasonable means of containing an intruder and monitoring his actions. This does not mean, however, that creating a jail is the best response to the presence of a hacker. As Cheswick points out, the security of the jail was far from convincing and they were somewhat surprised that the intruder seemed to either not care, or not know of its existence. Had either a more knowledgeable or determined intruder discovered that it was a jail, or possibly identify a path out of the jail, serious damage could have resulted. Had the circumstances of the Berferd attack been different, important data and network service may have been greatly effected. Before others begin to implement their own version of a jail, the expected goals and benefits must be carefully examined.

While they can be both interesting and entertaining, the knowledge gained from using a jail is often not worth the risk or effort.

17.3 The Internet Worm

In November, 1988 a Cornell graduate student released a computer worm that, within 48 hours, brought thousands of computers world-wide to a virtual standstill [17.8]. This worm began its journey from a Stanford Computer Laboratory and continuously duplicated itself onto networked computers [17.8]. As it traveled across the Internet, the worm not only reduced system performance but raised both concern that it may be malicious and confusion at its actions. Naturally, this led to widespread speculation and panic. Teams across the world set out to understand its functionality and determined how to stop it [17.8]. The ultimate goal being to completely eradicate the worm and fix any resulting damage.

At Berkeley and MIT, independent teams realized that the worm was migrating from system to system by taking advantage of common vulnerabilities in the Berkeley version of the UNIX operating system [17.8]. These vulnerabilities allowed an infected system to send a program to another system and instruct the receiving system to run the program. Once executed, the receiving system would download the worm from the infected system, thus infecting itself, and then search for other potential targets. This process was possible because of specific vulnerabilities in the *rsh* and *rexec*¹⁷ commands. In the event the first attempt was unsuccessful, the worm tried to exploit a vulnerability in the *finger* command, and finally a *sendmail* vulnerability [17.8].

The worm's first attempt to infect a system started with an attempt to locate and utilize trusted connections normally established for the *rsh* and *rexec* command. When a trusted *rexec* connection was successfully made, the worm infected the system. When the worm was unsuccessful the *finger* program was used.

While the *finger* program exists to provide a user information concerning other users, it also had the undesirable bug of interpreting its input as commands under certain conditions [17.3]. The worm would take advantage of this bug and force the target system to open a communications channel to the attacking system. The nature of this attack, however, was such that only VAXs running certain versions of the Berkeley UNIX operating system would be susceptible. Nonetheless, this did not deter the worm. Unsuccessful attacks against the *finger* program were followed by attacks against the *sendmail* command.

¹⁷ The *rexec* command functions much like the *rsh* command (see footnote 10). *Rexec*, however, requires the remote user to provide a password as it does not rely upon trusted hosts [17.5].

The vulnerability exploited in the *sendmail* program was similar to that exploited by Berferd on his attack of the AT&T Bell Labs computer system [17.2, 17.8]. When the debugging features of some versions of *sendmail* are enabled, a user can mail and execute a program on the receiving system. Unfortunately, early versions of the Berkeley UNIX and Sun Operating systems were susceptible and shipped with the *sendmail* debugging features enabled. This vulnerability, however, could have been easily removed by disabling the debugging option.

While work was being done to find and plug the vulnerabilities, teams were working to determine the purpose of the worm. Since unauthorized programs or activity often has malicious intent, and this program had been putting the computers into a catatonic state, it was reasonable to assume this worm had a malicious purpose. In an attempt to determine its purpose, people began to decompile the worm program [17.9]. Analysis of the worm, or any program for that matter, is a very arduous and time consuming process. The urgency to decompile the worm grew as researchers found instructions for the worm to perform an unknown task, named by its author as 'H_Clean', every twelve hours [17.6]. While the worm itself was dangerous, the 'H_Clean' procedure was determined to be no direct threat as it clean up some of the worms internal data.

Further analysis of the worm's code revealed significant flaws in the communications between multiple worm programs located on the same machine. This flaw led each instance of the worm to believe it was the first to infect a machine, and thus the system became inundated with numerous copies of the worm executing simultaneously. The significant computational burden of processing the worm programs was keeping each of the systems from perform any other task (hence the catatonic state). Aside from keeping a system from performing its daily tasks, the worm was harmless.

Once the worm's had been controlled and normal system operations could return, two questions were raised by the system owners and computing public. The first was whether the worm's code should be made public. The second question concerned the severity of the punishment the worm's author should receive.

Once the worm was completely analyzed, debate grew over the safety of making the program's source code public. Many people believed it would only encourage others to improve and emulate the worm. Others believed that release of the source code would be harmless because the vulnerabilities exploited by the worm were already being removed. Furthermore, they claimed, it was not their position to dictate to the public how the source code should be used. In the end, only details about the worm's operation were released.

Debate over an appropriate punishment for the worm's author was equal to, if not larger than, that concerning the release of the source code. Many people thought that the author should receive minimal or no punishment as there was no irreversible damage and the worm actually helped by identifying faults. Others argued that

regardless of the increased awareness that resulted, the author’s actions were deliberate and valuable computing time was lost. The only opinion that mattered, however, was that of the courts. In January, 1990 the court declared the worm’s author guilty of committing a federal fraud by unleashing a worm on the Internet under 18 U.S.C. § 1030 (a)(5)(a) [17.13]. For his crimes, the author received a three year probation, a \$10,000 fine, and four hundred hours of community service.

17.4 Adventures on the World Wide Web

The three incidents discussed above range anywhere from 5 to 10 years old. A lot has changed since then. When hacking and cracking was in its infancy, the big targets were the telephone companies and military bases. With the increased popularity in the Internet and all that is technical and bleeding edge, the targets have changed. Now, victims include large corporations, governments, and especially banks. The now-popular hacker clubs have also moved into expressing their political views and going after the “Goliath” organization’s systems to show their prowess. This is evident by the significant number of large-sized and well-known companies that have been vandalized on the World Wide Web. The relatively small table below, [Table 17.1](#), contains a time line of just a few (about 125) incidents that have occurred in the last 30 years. This information was compiled from many sources on the Internet (one of the best being Bill Wall’s security page). While the accuracy of these incidents is always in question and difficult to validate, their frequency is in no doubt growing.

Table 17.1. A partial chronology of known hacker and cracker activity in the last 30 years.

1971	Cap’n Crunch uses whistle (blue box) to access telephone company systems
1981	Kevin Mitnick broke into the records of Louisiana Unified School District, Monroe High School
May 1981	Kevin Mitnick gets into Pac Bell’s COSMOS phone center and steals passwords
1982	Kevin Mitnick cracks Pacific Telephone system and TRW and destroys data
1986	Chaos Computer Club cracks German government computer that had info about Chernobyl
September 1986	Stanford University computers hacked
September 1987	AT&T computers hacked
September 1987	Hackers from Brooklyn penetrate MILNET

November 1987	Chaos Computer Club hacks NASA's SPAN network
September 1988	Prophet cracks BellSouth computer network
November 1988	Internet worm sent out by Robert T. Morris
November 1988	US Air Force computer in San Antonio hacked
December 1988	Kevin Mitnick cracks MCI DEC network
June 1989	US Air Force satellite positioning satellite hacked
July 1989	Fry Guy cracks into MacDonald's mainframe; also stole credit cards
October 1989	WANK worm attacked SPAN VAX/VMS systems
March 1990	NASA computer at Huntsville and Greenbelt hacked
March 1990	Cliff Stoll's computer hacked
April 1990	Department of Defense sites hacked from the Netherlands
October 1990	British clearing banks hacked
January 1991	Lamprecht (Minor Threat) hacks into Southwestern Bell
April 1991	Dutch hackers from Eindhoven break into US military computers
1992	Eindhoven University of Technology in the Netherlands hacked
November 1992	Kevin Mitnick cracks into the California Department of Motor Vehicles
February 1994	Texas Racing Commission computer hacked
February 1994	Hacker spoofed a Dartmouth professor using email to cancel tests
July 1994	Pentagon Hacked
November 1994	FBI's conference-calling system hacked
December 1994	US Naval Academy computer system hacked
December 1994	Kevin Mitnick cracks into Tsutomu Shimomura's security computers
January 1995	Kevin Mitnick cracks into the Well
June 1995	US Air Force web site hacked through Vanderbilt computers
July 1995	Hackers tap into US Navy computer system and gain access to French and Allied data
July 1995	US military computers, Harvard, and NASA hacked
August 1995	New York Times Internet service hacked
September 1995	Berkeley students crack Tower Records/Video computers
December 1995	NASA Ames Research Center web site hacked
January 1996	Swedish computer hacker breaks into 911 phone system in Florida
January 1996	Chaos taps clear-text transmission of banking information
February 1996	Hackers altered United Kingdom talking bus stops for use to the blind
February 1996	BerkshireNet in MA hacked; data erased and system shut down
March 1996	Boston ISPs hacked by U4ea; deleted Boston Globe web pages
March 1996	Telia, Sweden's biggest ISP has web site hacked
April 1996	NYPD voice-mail system hacked
April 1996	Cambridge University hacked

June 1996	Public library network hacked
July 1996	Ontario hackers break into computers at a military base in Virginia
July 1996	High school students crack a drink manufacturer's computer voice-mail system
August 1996	Fort Bragg soldier compromised military computer system; distributed passwords
August 1996	European parliament and commission computers hacked from the US
August 1996	Department of Justice web site hacked
August 1996	American Psychoanalytic Association hacked
August 1996	Nation of Islam web site hacked
August 1996	British Conservative Party web site hacked
September 1996	Hackers shut down PANIX, New York's Public Access Networks
September 1996	CIA web site hacked
September 1996	Internet Chess Club hit by hacker attack
September 1996	Palisades, NJ school system hacked
September 1996	Cancelbot attacks Usenet; 25,000 messages wiped out
September 1996	Kevin Mitnick indicted for damaging computers at USC, stealing s/w
October 1996	Disgruntled employee wipes out all computer files at Digital Technologies Group
October 1996	Czechoslovakian banks hacked
October 1996	Florida Supreme Court web site hacked
November 1996	Anti-military site hacked
November 1996	New York Times web site hit with denial of service
November 1996	Latin Summer Meeting web site hacked
November 1996	Kriegsman furs web site hacked by animal rights activist
November 1996	Hackers removed songs from computers at U2's Dublin studio
November 1996	Disgruntled computer technician brings down Reuters trading network in Hong Kong
December 1996	British Labour Party web site hacked
December 1996	Approximately 3,000 WebCom web sites hacked
December 1996	Yale School of Medicine web site hacked
December 1996	NASA web site hacked
December 1996	US Air Force web site hacked at DTIC
December 1996	NASA web site hacked again
January 1997	California state agency computer hacked and crashed
January 1997	Crack dot Com hacked; Doom, Quake, and Golgotha source code downloaded
January 1997	Government of Victoria, Australia web site hacked

February 1997	Hackers spoof Eastern Avionics web site to obtain credit card numbers
February 1997	German Chaos group uses ActiveX and Quicken to withdraw money
February 1997	Indonesia's Department of Foreign Affairs web site hacked
March 1997	Loran International victim of a denial of service attack
March 1997	NASA web site hacked
March 1997	NCAA web site hacked by 14-year old
March 1997	Spammer Cyber Promotions suffers hack attack
April 1997	Malaysia's national telecommunications company web site hacked
April 1997	NASA web site hacked
April 1997	San Antonio's Express News web site server system hacked
April 1997	Cyber Promotions web site altered, password file stolen
April 1997	Amnesty International web site hacked
April 1997	British Conservative Party hacked
May 1997	The Lost World Jurassic Park web site hacked
May 1997	LAPD hacked
June 1997	USDA site hacked
June 1997	Hackers caused denial of service to Microsoft's NT IIS web server
June 1997	Geocities front page hacked
June 1997	Portuguese hackers attack Indonesian Government web site
July 1997	Canadian Security Intelligence Service hacked
July 1997	Swedish Crack-a-Mac web site hacked
July 1997	MacInTouch hacked
August 1997	George Mason University students hack their way into the University computers
August 1997	Cyber Promotions servers hacked
September 1997	Altavista web site hacked
September 1997	Coca-cola web site hacked
September 1997	Florida State School of Criminology server hacked
September 1997	US Geological Survey server hacked
September 1997	ValueJet web site hacked
October 1997	Hacker spoofs SANS Security Digest newsletter; breaks into ClarkNet ISP
October 1997	Japan's Nippon Telegraph and Telephone (NTT) hacked
October 1997	Yale e-mail account servers hacked
November 1997	Spice Girls web site hacked
November 1997	Numerous commercial and government web sites in Indonesia
January 1998	Unicef web site hacked
January 1998	Again, numerous commercial and government web sites in Indonesia
January 1998	The International Churches of Christ web site hacked

February 1998	US Department of Commerce web site hacked
September 1998	New York Times web site hacked
September 1998	Slashdot.org web site hacked
September 1998	Government of the city of San Pedro, Mexico web site hacked
October 1998	University of Texas at Brownsville web site hacked
October 1998	US Geological Survey web site hacked
October 1998	High resolution ink manufacturer web site hacked
October 1998	Chinese Government web site hacked
October 1998	US Army web site in Stuttgart, Germany hacked
October 1998	Wizard Communication Systems ISP hacked
November 1998	United Kingdom Cartoon Network hacked

There are three things the reader should take away from this table. First, it is by no means complete. Many organizations never make public the fact that they were attacked for fear of additional attacks and loss of customer trust—thus the word “known” in [Table 17.1](#)’s title. Second, the number of actual attacks would be much higher if those originating from outside the United States, particularly those in the late-1980s/early-1990s, were better documented and received more publicity. Third, this list does not show the number of arrests and other punishments that have resulted from this activity. While they are relatively few in comparison to the number of attacks, they do exist. Governments and companies are no longer taking attacks lightly; they are pushing the limits of the law and defining new law as it is needed. Finally, this information is not presented to glorify these types of activities. They are all serious crimes that are costing the victims, and sometimes the victim’s customers, a significant amount of time, effort, and money. As such, they should not be taken lightly.

17.5 Summary

This chapter presents three incidents in which computer security was breached. In each of these incidents, the principals responded differently and achieved different results. In one case, the system administrator attempted to track the intruder back to their point of origin. The administrator’s success at this endeavor required as much patience, determination, and hard work as that shown by the intruder. In the second case, the administrator attempts to place the intruder into a virtual jail with the intent to monitor the intruder and learn from his actions. While this task became rather involved, it did meet with some success, albeit minimal. It’s therefore suggested that this course of action be given as much thought and involve much pre-planning before it

is attempted. The third case did not involve chasing or tracking an intruder, but tracking the activity of a program. Fighting an inhuman opponent poses many different challenges not otherwise present in a human antagonist. For example, both the time and manner in which a computer reacts and responds is quite different from a person. Pre-programmed computer activity will be quick and consistent. These differences can be exploited in the protection and decomposition process. Regardless of whether the threat is human or not, it must be stopped.

17.6 Projects

- 17.1 Many of the problems found in the Hanover Hackers incident were directly related to the inactivity of system administrators. Create a set of system administration policies and procedures to address these issues.
- 17.2 On paper, design an electronic jail like the one used on Berferd. Detail the changes that would be required to the network services, applications, user accounts, logging features, and the operating system itself.
- 17.3 Implement the jail designed in Project 17.2. Detail how this process would be different for different operating systems such as Windows NT and the different variants of UNIX.

17.7 References

- 17.1 Adam, John A., "Threats and Countermeasures", *IEEE Spectrum*, IEEE/IEE Publications, Vol. 29, No. 8, August 1992, pp. 21-28.
- 17.2 Cheswick, William R., and Bellovin, Steven M., Firewalls and Internet Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1994.
- 17.3 Curry, David, UNIX System Security, Addison-Wesley Publishing Co., Reading, Massachusetts, 1992.
- 17.4 Eichin, Mark W. and Rochlis, Jon A., "With Microscope and Tweezers: An analysis of the Internet Virus of November 1988", *MIT Technical Report*, Massachusetts Institute of Technology, February 1989.

- 17.5 Ferbrache, David and Shearer, Gavin, UNIX Installation Security and Integrity, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
- 17.6 Frisch, Aileen, Essential System Administration, O'Reilly & Associates, Sebastopol, California, 1991.
- 17.7 Hafner, Katie and Markoff, John, Cyberpunk: Outlaws and Hackers on the Computer Frontier, Simon & Schuster Inc., New York, New York, 1991.
- 17.8 Seeley, Donn, "A Tour of the Worm", *University of Utah Technical Report*, The University of Utah, 1988.
- 17.9 Spafford, Eugene H., "The Internet Worm Program: An Analysis", *Purdue Technical Report CDS-TR-823*, Purdue University, **West Lafayette, Indiana, November 1988**.
- 17.10 Sterling, Bruce, The Hacker Crackdown: Law and Disorder on the Electric Frontier, Bantam Books, New York, New York, 1992.
- 17.11 Stevens, Richard W., TCP/IP Illustrated, Volume 1, Addison-Wesley Publishing Co., Reading, Massachusetts, 1994.
- 17.12 Stoll, Clifford, "Stalking the Wily Hacker", *Communications of the ACM*, Vol. 31, No. 5, May 1988, pp. 484-496.
- 17.13 West Federal Reporter Series, 928f.2d 504 (2d Cir. 1991), West Publishing Company, Saint Paul, Minnesota, 1991.

17.8 Extended Bibliography

- 17.14 Denning, Peter J. editor, Computers Under Attack: Intruders, Worms, and Viruses, Addison-Wesley, Reading, Massachusetts, 1990.
- 17.15 Hoffman, Lance J. editor, Rogue Programs: Viruses, Worms, and Trojan Horses, Van Nostrand Reinhold, New York, New York, 1990.
- 17.16 Levin, Richard B., The Computer Virus Handbook, McGraw-Hill, Berkeley, California, 1990.

- 17.17 Safford, David; Schales, Douglas and Hess, David “The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment”, *Proceedings of the Fourth USENIX Security Symposium*, Santa Clara, California, October 1993, pp. 91-118.
- 17.18 Sterling, Bruce, The Hacker Crackdown, Bantam Books, New York, New York, 1992.
- 17.19 Thompson, Ken, “Reflections on Trusting Trust”, *Communications of the ACM*, ACM Press, Vol. 27, No. 8, August 1984, pp. 761-763.

Appendix A

INFORMATION WARFARE

The term *Information Warfare* has become popular to describe what some believe to be the method for the next, or as a few believe, current, great global conflict. The battlefield for this conflict will not be found in the cities and countryside of any distant land but rather in the fields of silicon found in our computer systems. The weapons of choice will not be the bombs, bullets, tanks, and planes we have associated with warfare in this century but rather the keyboard, monitor, and telecommunication systems that permeate society. The assets being fought over will not be the lands, cities, factories, and natural resources of a country but simply information. Before one scoffs at the idea of information being both the object and the means of warfare, remember that nations increasingly depend on computers and the information they process to operate not only their telecommunications and financial services but their aircraft, cars, hospitals, businesses, and homes as well. Computers store information about us ranging from medical records to our past and current financial status. Of immense importance is how closely our computer and telecommunications industries are now intertwined. Our telephone system cannot operate without computers. At the same time our modern computer networks rely on our telecommunications base to provide reliable and inexpensive connectivity throughout the world. Our society has arrived at a point where both our businesses and governments rely on the computer and telecommunications industries to the extent that we no longer can imagine effectively operating without the products and services they provide.

Two thousand years ago Sun Tzu stated in The Art of War that skillful warriors watched for vulnerabilities in their opponents [A.15]. Through the many centuries that followed, military strategists have endeavored to follow this sage advice. If our reliance on computers and telecommunication facilities have become so dramatic then this is truly a vulnerability. If this in turn is true, then the question to ask is what exactly is the potential for the exploitation of this vulnerability? Computers at Risk, a 1991 report by the National Research Council lists six trends that underlie the assessment

that there is a growing potential for abuse of U.S. computer and communications systems [A.5]. The six trends listed are:

- Networks and the use of embedded computer systems are proliferating. This has greatly increased the number of individuals who now are dependent on computer systems, some without even realizing it.
- Computers have become an indispensable part of American business to the extent that computer risks can now also be considered general business risks (e.g., viruses).
- The proliferation of databases that contain highly personal information such as medical and credit records puts the privacy of every individual at risk.
- The increased use and dependence on computer systems in safety-critical applications increases the possibility that an attack on a computer system could result in the loss of human life.
- The ability of individuals to abuse computer systems is becoming more widespread and the attacks are often very sophisticated.
- The world political climate is unstable giving rise to the possibility of transnational attacks, especially at this time when the international nature of corporations and research is increasing.

These trends point to a number of vulnerabilities that we are becoming increasingly susceptible to. If we were to try and find a specific vulnerability, we have to look no further than the U.S. telecommunications network. Commercial carriers provide over 90 percent of the government's telecommunication services [A.1]. Since national security and emergency preparedness organizations in the U.S. rely so heavily on these telecommunication services, the government is greatly concerned with the possible threat to these public services [A.1]. During a time of national emergency, what additional damage would occur if these telecommunication services were also impaired? Intruders have already penetrated every element of these services including switching, transmission, and signaling systems as well as systems used for operation, administration and maintenance of the public packet switched network. One example of how these systems can be manipulated came to light in 1991 with the arrest of Kevin Poulsen. Allegations against Poulsen included charges that he modified existing telephone services, monitored telephone conversations, obtained unlisted telephone numbers along with the customer's name and address, added new telephone services, and forwarded calls to other numbers at will [A.1, A.16]. This example serves to show the extent that our telecommunication services can be manipulated and how exposed they actually are. To realize how vulnerable are we as a society, consider the following

quotation concerning our reliance on the National Information Infrastructure (NII) made at the 1996 Hearings on Security in Cyberspace:

Our communications, whether via telephones, fax machines, pagers, or cellular telephones increasingly rely on the NII as providers are replacing their analog switches with computer dependent digital switches. Much of the way money is accounted for, handled, and exchanged is now done via the NII. Salaries are directly deposited into bank accounts by electronic funds transfers. Automated teller machines ("ATMs") deposit funds, withdraw funds, and make payments. When payment is made for merchandise with debit cards and credit cards, transactions are verified using the public switched network. Much of our national economy also depends on the NII. The vast majority of transactions conducted by banks and other financial institutions are done via electronic funds transfers. For example, on major bank transfers approximately \$600 billion electronically per day to the Federal Reserve. Over \$2 trillion is sent in international wire transfers every day. In addition, most securities transactions are conducted via computerized systems. Health care is increasingly becoming dependent on electronic records as pharmacies and hospitals maintain computerized files containing their patients' medical profiles. Medical care is moving toward greater dependency on computer-based technologies; hospitals are testing the viability of "on line" remote medical diagnosis. Our civil aeronautics industry is reliant upon computers to fly and land airplanes; railway transportation is dependent upon computers to coordinate tracks and routes. Government operations are also heavily dependent on the NII. The government uses computerized systems to do everything from issuing Social Security checks to keeping track of criminal records. Within our national defense structure, over 95% of the military's communications utilize the public switched network. Many of the military's "precision" weapons depend on the Global Positioning System (the "GPS") for guidance. In addition, the military uses computerized systems to transmit data and information related to troop movements, procurement, maintenance and supplies [A.22].

A.1 Levels of Information Warfare

Information Warfare is not just an issue for governments to be concerned with. Just like any global conflict, this war will affect many more individuals than the combatants themselves. In addition, just like other forms of warfare, there are several different levels that this war may occur on. Winn Schwartau has described three

different levels of intensity with different goals, methods, and targets for each [A.14]. These three levels are:

- ***Information Warfare*** on a Personal Level
- ***Information Warfare*** on a Corporate Level
- ***Information Warfare*** on a Political Level

At the Personal Level, the attack is against a specific individual. The attacker may be engaging in this warfare for a number of reasons; revenge, financial gain, mischief, or for some other darker motive. An attack at this level may be directed at the records that define who we are in society or it may be aimed at the lifelines that connect us to the world such as our phone or other utilities. An example of an attack at this level occurred in 1984 to Richard Sandza, a writer for Newsweek magazine. Sandza had written an article which appeared in Newsweek entitled “The Night of the Hackers.” Certain individuals didn’t like what he wrote and Sandza subsequently found himself the target of a personal information attack. These individuals obtained his credit card account numbers and posted them along with his name and address on an electronic bulletin board [A.8]. Sandza had not been physically attacked or harmed but information about him was used as a weapon against him.

Another example of an attack at the personal level occurred numerous times in the case of Kevin Mitnick. For a decade, Mitnick, a ‘hacker’ and ‘phone-phreak’, manipulated numerous computers and phone systems to gain access to information on still other systems. (For the story of Mitnick’s early career in ‘hacking’ and ‘phone phreaking’, see [A.3].) When he became upset with an individual, they were likely to find their phone service disconnected or rerouted. He once, for example, manipulated the phone company’s billing so that a hospital’s \$30,000 phone bill was attached to the account of an individual Mitnick disliked [A.3]. His ability to manipulate the phone system was so overwhelming that when he was arrested in 1988 for breaking into supposedly secure computer systems and stealing software, the judge in the case denied him bail and severely restricted his telephone access, allowing only numbers approved by the court. Mitnick later was convicted and spent a short time in jail. In 1995, only a few years after he was released, Kevin was again in trouble with the law and arrested. This time he was accused of breaking into computers, illegally using cellular telephone services, and stealing thousands of credit-card numbers from an Internet service provider.

The target of warfare at a corporate level is a business. The target may be information stored on corporate computer systems needed to perform daily operations or it may be the image or reputation of the company which can be affected in numerous ways. In 1988, for example, Kevin Mitnick was turned down at the last minute for a job as a security consultant at the Security Pacific Bank in California when a tip to the bank revealed his previous criminal activities. Two weeks later a press release was sent

over a news service wire to a news service in San Francisco stating that the bank had experienced a first quarter earning loss of \$400 million dollars. Fortunately for the bank, a reporter called to confirm the report which was in fact totally false. The bank later estimated that the potential damage from falling stock prices and closed accounts could have easily exceeded the \$400 million figure had the release become public [A.6]. Though it was never proven (in fact there was no way to prove it short of a confession by the culprit) Mitnick was the prime suspect in this incident.

Another example of corporate vulnerability to information warfare also occurred in 1988. In September of that year, Gene Burleson, a former employee of USPA & IRA, a Fort Worth based insurance and investment company, was convicted of planting a 'time bomb' program which had destroyed 168,000 sales commission records [A.11]. Burleson had been having numerous conflicts with supervisors and was eventually fired from the company. A short while after his dismissal, the 'time bomb' went off destroying the records. Though he tried to cover his tracks, he was discovered and arrested. While both of these examples involve a single individual attacking a company or institution, it is important to note that corporate warfare could easily involve two companies. The attacks may be similar to the ones described or they could involve one company attempting to gain knowledge of corporate secrets to gain a competitive edge. An example of corporate level information warfare between two corporations can be seen in a court case involving Borland International, Inc. and Symantec Corp. In this case, an employee of Borland was accused of sending sensitive corporate documents to Symantec via the Internet. The employee had allegedly done this after agreeing to work for Symantec [A.7]. The perpetrator of corporate information warfare does not have to be other corporations or disgruntled individuals. We are seeing an increasing amount of government sponsored corporate espionage. In one case, an employee of Boeing was recruited by Russian intelligence agents while in the midst of a messy divorce which occurred before he was employed at Boeing. When the divorce was final, the individual was encouraged by the Russians to apply for a job at Boeing to "start a new life." Later, the Russians paid him to obtain information about the Strategic Defense Initiative (the "Star Wars" missile defense system). The individual obtained information by breaking into Boeing corporate computer systems using techniques taught to him by a computer expert in the GRU--the Russian military intelligence agency [A.22]. In another example of government sponsored corporate information warfare, several biotechnology firms in the United States were contacted by representatives of the NHK (the Japanese version of the PBS) ostensibly wanting to film a documentary on the U.S. biotechnology industry. The questions they asked and the shots they filmed, however, eventually convinced officials that they were doing more than just filming a documentary and that they in fact were on a corporate espionage mission--possibly at the direction of the Japanese External Trade Relations Office (JETRO) which many believe to be nothing more than a government sponsored corporate intelligence gathering organization [A.22].

No discussion of corporate information warfare would be complete without a comment on the introduction of organized crime into the information arena. Some government estimates place the amount of money being stolen from U.S. and European banks, security firms, and corporations at over \$1 million per day [A.1]. Because of the concern that they would lose customers if this type of information became public knowledge, few intrusions into banks have ever been acknowledged. One case that did become widely publicized occurred in 1994 and involved Citibank and a group of Russian 'hackers' led by a system administrator in St Petersburg, Russia. This group was able to break into Citibank's electronic money transfer system and steal over \$10 million. Fortunately for Citibank the theft was discovered and all but a few hundred thousand dollars was recovered [A.1, A.22].

Warfare at the political, or international, level involves a country as the victim. The attacker may be an individual acting on behalf of another country, or it may be a terrorist organization trying to gain publicity for their cause. The target may be government secrets, the military industrial complex, or the citizens. Probably the best known example of warfare at this level was the case of the West German spy ring which broke into U.S. computer systems and then sold the information they obtained to the KGB [A.6, A.18]. The ring, which was eventually broken in 1989 due mostly to the efforts of Clifford Stoll, an astronomer at the Lawrence Berkeley Labs, had gained access to numerous government computers throughout the world. It took Stoll several months to track the intruders due to the maze of systems they used to hide their trail. When finally caught, it was learned that the intruders had been selling the information obtained to the KGB for tens of thousands of dollars and small amounts of cocaine.

Another example of the potential harm that can occur to governments from *Information Warfare* occurred during the summer of 1994. A Scottish 'hacker' was able to break into British Telecom databases gaining access to thousands of pages of confidential records [A.5]. The details of these records, which were subsequently published in London's newspapers, included the phone numbers for the Prime Minister and Royal Family, as well as the location of certain secret military installations and secret buildings used by the British intelligence services [A.5]. While the individual who perpetrated this break-in did not intend to cause any harm, the same information could have been obtained by a terrorist organization whose methods and goals are much more destructive.

The potential of information warfare during an actual conflict was demonstrated extremely well in the Gulf War. During the opening minutes of the conflict in January, 1991, the U.S. and coalition forces primary target was the Iraqi command and control structure. Command posts, headquarters, and power and telephone centers were destroyed effectively leaving the Iraqi military and government deaf, dumb, and blind. The highly centralized Iraqi command and control system was effectively eliminated by targeting electrical power and telecommunication centers with U.S. Navy Tomahawk

cruise missiles and precision-guided munitions dropped from U.S. Air Force F-117 Stealth Fighters. Without their information processing assets, the Iraqi's did not have the information or means to communicate necessary to conduct an effective defense against the coalition air campaign [A.2].

A final example of the potential harm that a country can suffer should its information systems be targeted occurred in November of 1988. At that time a graduate student at Cornell University, Robert Morris, released what has come to be known as the *Internet Worm*. Though it appears that the resulting confusion was unintentional, the worm caused several thousand machines to grind to a halt [A.6, A.18, A.19]. Several days were spent attempting to rid the network of all traces of the worm, with numerous sites simply 'unplugging' themselves from the network to avoid any possible harm. The loss in terms of the hours spent in cleaning up after the worm, as well as the lost hours of work, has been a subject of debate with some figures placing the figure well over a million dollars. The figure that was used in court during the trial of Morris was \$160,000, a number which the public could more easily understand. While it appears that Morris' intentions were not to bring the Internet down, this was the result. What are the implications of this worm if we assume that there are organizations that may indeed want to disrupt the research efforts of numerous companies, universities, and government agencies? What would have been the result had a terrorist organization performed the act instead?

A.2 Weapons of Information Warfare

There are many weapons used in a conventional war. *Information Warfare* also may employ many different weapons. Obviously one of these weapons is the computer itself. An individual such as Kevin Mitnick, Kevin Poulsen, or Robert Morris sitting at one of these machines can be the cause of much damage to computer systems and the data they process. Winn Schwartau has described a number of different weapons which can be used to conduct information warfare [A.15]. Among these are:

- Malicious software
- Sniffers
- Electromagnetic eavesdropping equipment
- Computer chips
- HERF guns
- EMPT bombs

We are already aware of the damage that can be caused by malicious software such as viruses, time bombs, worms, and trojan horses. Usually we think of this type of

software as being developed by a disgruntled employee or college students with a bit too much time on their hands. Usually this type of software ‘announces’ its presence when it performs its destructive action, leaving no doubt that the system and user have been the victim of some form of malicious attack. What if, instead, the malicious software was carefully designed by professionals with the intent of damaging a specific corporation? A virus could be written, for example, to only attach itself to a specific piece of software, such as a spreadsheet or database management system from a certain vendor. It could also be written in such a manner that it did not activate its destructive segment until a long latency period had elapsed allowing it time to infect a large number of systems. Then, when it did activate, instead of doing something extremely visible, such as reformatting a disk, it might just change a few bits here and there. Nothing major, just enough to cause the user to become frustrated with the software package. Eventually, the users of the software would become so disenchanted that they would stop using it and might instead purchase a competitor’s package. Malicious software designed and engineered by professionals, instead of the unorganized activity of current “hobbyists”, would have an even greater impact if the target was the software that controls our networks and computer systems themselves. We have only just begun to see the potential impact malicious software can have.

Sniffers are used to ‘sniff’ or monitor a network. They listen to the traffic that is sent between systems in a network searching for specific packets or types of packets. Often the goal of a sniffer is to obtain userids and passwords but they can also be used to garner unencrypted mail or files sent between users or even to grab and modify files before they arrive at their destination. The widespread use of encryption can reduce the impact of this form of attack.

Electromagnetic eavesdropping, also known as TEMPEST, (Transient Electromagnetic Pulse Emanation Standard) is designed to take advantage of the fact that electronic equipment emits signals that can be picked up with specialized equipment. This means that an individual can sit in one room and pick up the information being displayed on the terminal or printer in the next room. This brand of information warfare can be used to collect data files as they are being displayed, or even passwords and userids to be exploited later. Shielding the computer equipment can help protect against this form of warfare as can shielding the entire room in which the equipment resides (although this is a very expensive prospect). This type of attack has also been referred to as “van Eck phreaking” after Win van Eck, a scientist in the Netherlands, demonstrated its potential [A.7].

“Chipping” is the hardware version of malicious software. It involves designing computer chips which contain back doors or other hidden features not known to the users of the chips. These special chips might even be designed to display a specific electromagnetic footprint or signature which might allow other equipment (or weapons) to locate or home in on them. (Some have speculated that this type of information warfare was employed by the United States against Iraq during the Gulf War [A.15].)

High Energy Radio-Frequency (HERF) guns are designed to exploit the same effect used by TEMPEST equipment except in an offensive manner. These weapons are designed to emit a burst of high energy at a high frequency in order to overload the circuits of computers and other electronic equipment. These devices have reportedly been able to disrupt the operation of computer systems at ranges up to 300 yards [A.7, A.15]. We have all probably seen this phenomenon in operation without realizing its information warfare potential. How often have we noticed that certain pieces of electrical equipment has interfered with our televisions, or how often have we had our radio station reception interfered with as we drove near high power cables? The same phenomenon, employed by terrorists in a van packed with HERF type weapons, could be used to disrupt the operation of thousands of computer systems and consequently adversely affect numerous companies if the weapon was employed in an area such as Wall Street [A.7].

Electromagnetic Pulse Transformer (EMPT) bombs take advantage of technology learned in the design and testing of nuclear weapons. These bombs are designed to fuse the chips of our computer systems and other electronic equipment together with a high powered electromagnetic pulse. The possibility of this form of attack has been known for a number of years as is evidenced by the U.S. military's efforts to shield its most important equipment from the possible effects of a nuclear weapon.

A.3 Perception Management

Up to this point the discussion of Information Warfare has centered on the technical aspects of this type of conflict. One other side of information warfare, made even more effective because of technology today, is the ability to shape people's perception of events in order to affect their decisions and actions. With the existence of stations such as the Cable News Network (CNN), people today can tune in and view live footage of events as they occur. The ability to know what is happening around the world the instant it occurs is not inherently a bad thing—provided that the reporting of the incidents is done accurately and without bias. This, unfortunately, is not always the case and unscrupulous leaders can affect the perception of events by shaping events specifically for international news organizations. This has prompted some to refer to global television as the “Poor Man's Command and Control Warfare” [A.3]. Consider, for example, the following quote from Chuck de Caro, a former Special Assignments Correspondent for CNN and the President and CEO of Aerobureau Corporation:

A tenth-rate tin-pot Haitian dictator ... judged the likely U.S. reaction in the wake of revulsion at the video-tape of Rangers being killed and mutilated in Somalia. He optimized his political-military moves to

forestall U.S. intervention by having a handful of rabble assemble on a pier, mug angrily-on-cue for global TV while waving English-language placards. He thus turned away a U.S. warship-on a U.N. mission-with nothing more than the video of an alleged mob that generated the perception of imminent bloodshed projected and amplified by TV. The perception was worsened by video coverage of the warship sailing away [A.3].

Another excellent example of shaping public opinion in order to affect the actions of a group occurred shortly after the Iraqi invasion of Kuwait. Many people remember the testimony before a congressional committee by a 15 year-old Kuwaiti girl known only as “Nayirah” (the rest of her name ostensibly being kept a secret to protect family and friends in Kuwait). Her testimony included the following:

I volunteered at the al-Addan hospital . . . while I was there, I saw the Iraqi soldiers come into the hospital with guns, and go into the room where 15 babies were in incubators. They took the babies out of the incubators, took the incubators, and left the babies on the cold floor to die [A.9].

Upon hearing this the American public was outraged. President Bush mentioned this incident numerous times in the weeks to follow as he attempted to rally public support for military action to be taken to free Kuwait. After this testimony several Kuwaiti doctors came forward to testify that they too had witnessed similar events. This single event did more to build public support for the president than any other event. The only problem with it is that it now appears that it was all fabricated by a public relations firm hired by the president of Citizens for a free Kuwait, Dr. al-Ebraheem [A.9, A.17]. The public relations firm, Hill and Knowlton, is one of the largest and best connected in America. Nayirah, as it turned out, was actually the daughter of Saud al-Sabah, the Kuwaiti ambassador to the U.S. The doctors who testified they were also witnesses to similar events could never provide any proof. In fact, one of them, who had given his name as Dr. Issah Ibrahim, was later discovered to be Dr. Ibrahim Behbehani and was not actually a surgeon but a dentist [A.9]. After the war he admitted “I can’t tell you if they [the babies] were taken from incubators...I didn’t see it.” [A.9]. John Martin, of ABC News, tried to verify the reports of the incubator atrocities interviewed Dr. Fahima Khafaji a pediatrician at the hospital where Nayirah had supposedly witnessed the events she described. When asked if she could verify the story, Dr Khafaji replied that she had not witnessed any such events [A.9]. Martin also interviewed Dr. Mohammed Matar who was the director of Kuwait’s primary health-care system and his wife Dr. Fayeza Youssef who was the chief of obstetrics at the maternity hospital. Dr Youssef stated that “No, [the Iraqis] didn’t take [the babies] away from their incubator...” [A.9]. After numerous interviews Martin

concluded that people, including babies did indeed die “when large numbers of Kuwait’s doctors and nurses stopped working or fled the country in terror” [A.9]. The testimonies of these Kuwait doctors after the war are further corroborated when compared with the testimony of an Icelandic doctor, Dr. Gisli Sigurdsson) who had left Kuwait in November and stated that, on the subject of the incubator story, “That news was not true... However, there were lots of babies who died because of lack of staff over the last few weeks.” [A.9]. It is unclear how many individuals at the time, including the president, knew that the incubator story was most likely a complete fabrication. At the time it didn’t seem important to check on the details since the incident was in keeping with what most people wanted to believe about Iraq. What is important to note from an Information Warfare standpoint is how effectively this campaign worked and how it might serve as an example of perception management in the future.

A.4 Summary

The widespread use of computer systems and networks in our society has led us to a point where the information they contain, and the services they offer, can be used as weapons against us. This is not only true for us on a personal basis, but is true for corporations protecting their industrial and trade secrets, as well as countries with their military and diplomatic secrets. The weapons and warriors of this new type of warfare include programmers working on computer systems, terrorists with HERF guns, and nations employing EMPT bombs.

All of this leads us to the question of what is to be done about the possibility of information warfare? Should we be considering the establishment of what amounts to CyberCops as we expand the power of our law enforcement agencies in cyberspace [A.14] or instead should an awareness program be initiated to insure that the dangers of information warfare are known and understood by everyone? In reality, the answer is probably a little of both. Finally, what is to be done about the media’s ability to shape the perception of events and consequently the actions that we might take? It is incumbent on the news media to attempt to supply an unbiased and factual report to the public. At the same time, it is important that the public realizes that mistakes can be made and the media can be exploited by individuals to promote their own agenda. As such, we must always be wary of reports that can not be easily substantiated and we must attempt to not let our own preconceived notions about others influence our perceptions.

A.5 Projects

- A.1 Assume the role of an *Information Warrior*. List a number of targets that would be available to you in a Personal Level conflict.
- A.2 Again assume the role of an *Information Warrior* but this time determine targets for a Corporate Level conflict.
- A.3 Once again assume the role of an *Information Warrior* terrorist now intent on carrying out warfare on the International or Political Level. What are your targets now?
- A.4 From a military perspective, what Information Warfare targets exist whose destruction or disruption could affect an enemy's fighting capabilities while enhancing yours (consider U.S. and Coalition Force examples during the Gulf War)?
- A.5 How could "chipping" be used to insert covert channels into a computer system?
- A.6 Consider the infamous bombing of the World Trade Center. Economic losses due to the disruption of business communication exceeded the actual physical damage to the building. What implications does this have for *Information Warfare* ?

A.6 References

- A.1 Air Force Information Warfare Center Computer Threat Advisory, CTA 95-027, September 1995.
- A.2 Campen, Alan D., "Iraqi Command and Control: The Information Differential" The First Information War, AFCEA International Press, Fairfax, Virginia, 1992.
- A.3 de Caro, Chuck, "Softwar", Cyberwar: Security, Strategy and Conflict in the Information Age, AFCEA International Press, Fairfax, VA, 1996.

- A.4 Frizzell, Joseph; Phillips, Ted and Groover, Traigh, "The Electronic Intrusion Threat To National Security & Emergency Preparedness Telecommunications: An Awareness Document", *Proceedings of the 17th National Computer Security Conference*, Baltimore, Maryland, October 1994, pp. 378-388.
- A.5 Goldstein, Emmanuel, "Inspiration", *2600 The Hacker Quarterly*, Winter 1994-95, Vol. 11, No. 4, pp. 4-5.
- A.6 Hafner, Katie and John Markoff, Cyberpunk: Outlaws and Hackers on the Computer Frontier, Simon & Schuster, New York, New York, 1991.
- A.7 Kabay, M.E., "Prepare Yourself for Information Warfare", *Computerworld*, Leadership Series, March 20, 1995, pp. 1-7.
- A.8 Littman, Jonathan, The Watchman: The Twisted Life and Crimes of Serial Hacker Kevin Poulsen, Little, Brown and Company, Boston, Massachusetts, 1997.
- A.9 Macarthur, John R., Second Front: Censorship and Propaganda in the Gulf War, Hill and Wang, New York, NY, 1992.
- A.10 National Research Council, Computers at Risk: Safe Computing In the Information Age, National Academy Press, Washington, DC, 1991.
- A.11 Reese, Lloyd, "Computer Crime and Espionage: Similarities and Lessons Learned", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 389-395.
- A.12 Sandberg, Jared, "Undetected Theft of Credit-Card Data Raises Concern About On-Line Security", *The Wall Street Journal*, February 17, 1995, pg. B2.
- A.13 Sandza, Richard, "The Revenge of the Hackers", *Newsweek*, December 10, 1984, pg. 81.
- A.14 Schwartau, Winn, "Information Warfare:TM Waging and Winning Conflict in Cyberspace, An Outline for a National Information Policy", *Proceedings of the IFIP Sec'94 Conference*, Curacao, June 1994.

- A.15 Schwartau, Winn, "Information Warfare", Keynote Address, *IFIP Sec '94 Conference*, Curacao, June 27, 1994.
- A.16 Seline, Christopher J., "Eavesdropping on the Electromagnetic Emanations of Digital Equipment: The Laws of Canada, England and the United States", 1990 Draft Document, available via anonymous *ftp* at *ftp.funet.fi, /pub/doc/security/pyrite.rutgers.edu/tempest.cjs*, March 24, 1995.
- A.17 Stauber, John and Rampton, Sheldon, Toxic Sludge Is Good For You: Lies, Damn Lies, and the Public Relations Industry, Common Courage Press, Monroe, Maine, 1995.
- A.18 Stoll, Clifford, The Cuckoo's Egg, Doubleday, New York, NewYork, 1989.
- A.19 Stoll, Cliff, "An Epidemiology of Viruses & Network Worms", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 369-377
- A.20 Sussman, Vic, "Policing Cyberspace", *U.S. News and World Report*, January 23, 1995, pp. 54-60.
- A.21 Tzu, Sun, The Art of War, translated by Thomas Cleary, Boston, Massachusetts, 1988.
- A.22 Winkler, Ira, Corporate Espionage, Prima Publishing, Rocklin, California, 1997.

A.7 Extended Bibliography

- A.23 Adams, James, The Next World War: Computers are the Weapons and the Front Line is Everywhere, Simon and Schuster, New York, New York, 1998.
- A.24 Brewin. Bob and Sikorovsky, Elizabeth, "Information Warfare: DISA stings uncover computer security flaws", *Federal Computer Week*, February 6, 1995, pp. 1, 45.

- A.25 Carpenter, Ted Galen, The Captive Press: Foreign Policy Crises and the First Amendment, CATO Institute, Washington D.C., 1995.
- A.26 Denning, Dorothy, "Concerning Hackers Who Break into Computer Systems", *Proceedings of the 13th National Computer Security Conference*, Washington, DC, October 1990, pp. 653-664.
- A.27 Fialka, John J., War by Other Means: Economic Espionage in America, W. W. Norton and Company, New York, NY, 1997.
- A.28 Green, James and Sisson, Patricia, "The 'Father Christmas Worm'", *Proceedings of the 12th National Computer Security Conference*, Baltimore, Maryland, October 1989, pp. 359-368.
- A.29 Grier, Peter, "Information Warfare", *Air Force Magazine*, March 1995, pp. 34-37.
- A.30 Holstein, William J., "Corporate Spy Wars", *U.S. News and World Report*, Feb 23, 1998, pp. 46-52.
- A.31 Kabay, Michael, "Information Warfare Could Be More Than Fiction", *Network World*, September 6, 1993, pg.32.
- A.32 McDonald, Chris, "Computer Security Bloopers, Bleeps, Blunders", *Summary of Papers Presented at the Tenth DOE Computer Security Group Conference*, Albuquerque, New Mexico, May 1987, pp. 35-45.
- A.33 Schwartau, Winn, "Hackers indicted for infiltrating corporate networks", *INFOWORLD*, July 27, 1992, pg. 56.
- A.34 Schwartau, Winn, Information Warfare: Chaos on the Electronic Superhighway, Thunder's Mouth Press, New York, New York, 1994.
- A.35 Smith, Julie, "No Harm Intended: A Behavioral Analysis of Young Hackers", *Proceedings of the 8th National Computer Security Conference*, Gaithersburg, Maryland, October 1985, pp. 36-42.
- A.36 Sterling, Bruce, The Hacker crackdown: Law and Disorder on the Electronic Frontier, Bantam, New York, New York, 1992.

- A.37 Watts, Jim, "Computer-Related Fraud in the Banking and Insurance Industries", *Proceedings of the 7th DOD/NBS Computer Security Conference*, Gaithersburg, Maryland, September 1984, pp. 207-213.

Appendix **B**

UNIX SECURITY

Thanks to several widely publicized incidents, when most people hear the name UNIX it is generally in relationship to a security incident. UNIX, however, has been the operating system of choice on the Internet for a number of years. While the introduction of new PC-based operating systems may be threatening this once near-monopoly, UNIX and its many variants are a presence we must deal with if we are to secure the INTERNET. This appendix addresses this rather unique operating system, tracing its origin and growth and describing its various components in order to provide a picture of what UNIX is and what needs to be done to secure a UNIX-based system.

B.1 History

The history of UNIX began in 1965 when AT&T joined the MULTICS (MULTiplexed Information and Computing Service) research effort funded by the Department of Defense Advanced Research Projects Agency. AT&T joined other researchers from Honeywell, General Electric, and the Massachusetts Institute of Technology in an effort to design an operating system that could provide simultaneous access to multiple users, the ability for users to easily share data, and adequate computing power and storage space. MULTICS, since funded by the military, was also designed with the idea of providing a multi-level security environment in which users operating at various security levels would be protected/prevented from interfering with other users.

In 1969, with the MULTICS project behind schedule, AT&T dropped out of the joint effort. A number of researchers who had been involved on the project, however, still desired an operating system which could provide a friendly program design environment. Ken Thompson, using an unused PDP-7, pursued this goal along with Dennis Ritchie, also a former member of the MULTICS project. Together they worked

on what became known as UNIX—a play on the term MULTICS (which tried to do many things whereas UNIX was designed to provide one thing, a good programming environment to run programs). It is interesting to note in hindsight that strong security was not a priority requirement of this early system. In 1971 Thompson and Ritchie with help from other developers rewrote this now operational operating system for the new PDP-11. The system provided 8K bytes for the users, 16K bytes for the system itself, a disk of 512K bytes and had a limit of 64K bytes for files.

With an operational version of UNIX, Thompson now set out to develop a FORTRAN compiler for the system. What he created instead was an interpretive language called B (influenced by BCPL). This language quickly evolved into another language called C. In 1973 UNIX was rewritten in this new language, a move that had tremendous implications later.

Due to a 1956 Consent Decree, AT&T was forbidden from marketing computer products. The decree did not, however, stop AT&T from providing copies of UNIX free to universities for educational purposes. As more universities requested and received copies of UNIX the popularity of this new operating system grew. The interest in UNIX increased even further when in 1974 Thompson and Ritchie published a paper about it in the *Communications of the ACM*. The paper helped to further advertise the operating system to its readers mostly academics and computer professionals. By 1977 the number of sites using UNIX had subsequently grown to around 500.

When UNIX was distributed to users it included the source code for the operating system, written in C. Providing the actual source code for the system, especially since it was written in a higher level language, provided programmers with the opportunity to tailor the system as they saw fit. Many developers consequently modified the original system with their own enhancements. Ultimately two main versions surfaced; AT&T's (which went through several releases with formal support for their System V being announced in 1983) and the University of California at Berkeley's known as BSD (Berkeley Software Distribution) with several releases of its own. By 1984 almost 100,000 installations worldwide were using a version of UNIX with more systems being added every day. Today several other versions of UNIX may also be encountered including Sun Microsystems' version as well as XENIX (Microsoft), ULTRIX (DEC), and AIX (IBM). In 1991 the currently popular PC version, LINUX, was developed.

B.2 System Boot

Generally speaking, 'booting' the system will be done by an individual who has physical access to the machine. It is important to note that for many versions of UNIX (especially LINUX), if an individual can gain physical access to the machine then it is a fairly easy matter for the individual to gain root access to the system (through the use

of a special boot disk or accessing the system using single-user mode on machines that don't require a login in this mode of operation). This is an important point that can't be overemphasized.

All versions of UNIX store important information needed by the system to properly boot and configure itself in several files, scripts, and programs located at various places in the system. When the system is turned on UNIX will run the *boot* program which loads the system kernel (the most basic part of the operating system) into the computer. The *boot* program in turn will execute the *init* program found in the */etc* directory. This process receives the process id (PID) 1. Some systems are able to operate in single-user mode and will run a standard shell (such as *sh* found in the */bin* directory) on the console.

The next file UNIX will normally execute is the */etc/rc* script. This file may in turn, depending on the configuration of the system and the version of UNIX, execute several other scripts found in the */etc/init.d* or */etc/rc.d* directory. On AT&T System V versions the system will consult the file */etc/inittab* to identify what it should do at various levels of operation. Finally UNIX will fork (initiate) a new process (*/etc/getty* or */usr/lib/saf/ttymon*) for each 'terminal' on the system. It is this last process which is responsible for displaying the login prompt and communicating with the user.

Depending on the shell users specify to be their default, one of several different startup files will be consulted when a user logs in. These files include the *.profile*, *.login*, *.kshrc*, and *.cshrc* files. Each of these allow users to tailor their environment in certain ways.

B.3 Audit Features

The purpose of auditing is to keep track of what has happened on the system. An *audit trail* will contain a history of the commands that have been executed or specific actions that are taken. This will make it easier to track down problems that occurred or determine the severity of intrusions into the system. It must be noted that because audit trails are generally contained on the system itself they are a prime target for intruders wishing to hide their actions. On UNIX-based systems there are several files used for auditing (also known as logging and thus log files). Depending on the variant of UNIX these files may be found in one of several locations.

The first and most basic audit file is the *lastlog* file which maintains a record of the last successful (and in some cases the last unsuccessful) login for each user. This file in turn is checked at login in order to display these times. The theory is that a careful user will take note of the last recorded time and if it doesn't coincide with the last time the user knows he/she logged in then it is an indication of a possible

intrusion. The problem with this of course is that most users rarely actually take note of the time that is displayed. Another problem with this file is that it only keeps track of the last recorded login so repeated attempts over a period of time will not be noticed, just the last. The *lastlog* file is generally found in the */var/adm*, */var/log* or */usr/adm* directory (or a subdirectory beneath it).

The next files to discuss are the *utmp* and *wtmp* files. The *utmp* file is found in the */etc* directory and the *wtmp* file is found in the */var/adm* or */etc* directory. Current logins are recorded in the *utmp* file which contains an entry for each active tty session. Commands such as *who* and *users* extract information from this file. Just like the *lastlog* file, the *utmp* file only keeps a limited amount of information. The *wtmp* file on the other hand maintains an entry for every login and logout. This file, as can be imagined, can grow to a considerable size if it is never archived and cleared. The *last* function can be used to display the contents of the *wtmp* file in a nicely formatted manner. Be aware, however, that this will display the entire file.

A related and useful file on some versions of UNIX is the *su*log which keeps a record of all *su* login attempts. Since gaining superuser access is often an immediate goal of intruders, this file can play an important role in determining when an intrusion has occurred. Periodically scanning this file can reveal users (or intruders who have gained access to an authorized user's account) attempts to gain increased privileges. Unfortunately if an individual does gain superuser privileges this file often becomes a target itself. For this reason some system administrators will run programs which automatically send entries in this file to either a printer or another secure system on the network.

The files discussed so far can provide indications of intrusions but what actions did an intruder take once access was obtained? None of the files mentioned above track the commands that an individual executed. For this level of detail the *acct* or *pacct* file may be used by running either the *lastcomm* or *acctcom* program. These files will contain a record of all actions taken by users of the system. As can be imagined, recording all actions for all users will result in an extremely large file. For this reason this level of accounting is not automatically invoked when the system is booted. Depending on your version of UNIX, either the *startup* or *accton* program must be run to initiate this level of auditing. As there are a number of options for these programs, the manual pages should be consulted to insure the desired auditing is obtained.

The final log file we will mention is the UNIX *syslog* facility found on many versions of the operating system. This facility, originally designed for use with the *sendmail* program, provides a general purpose logging feature for messages which can now be generated by any program. The actual location of where the messages will be stored is specified in the *syslog.conf* configuration file. The *syslog* facility has grown to offer quite a few options and the manual pages should be consulted to ensure the proper use of this logging facility.

B.4 Passwords and Accounts

The first line of defense in UNIX security is the password file. This is also the area that causes the most problems for security administrators. In UNIX-based systems the passwords can be found in the */etc/passwd* file. This location and the format for this file is well-known. Each line of the password file contains information used by the system during the login process for the user and various functions will allow access to the fields in this file. A sample UNIX *passwd* file might look like the following:

```
root:a39Sgk2k33QC3:0:0:Sysadmin:./bin/sh
uucp:l94GAX3iq3s2s:4:4:uucp:/var/spool/uucppublic:/usr/lib/uucp/uucico
mickey:983al2di23IJ3:200:200:Mickey's acct./user/home/mickey:/bin/csh
minnie:0293jfjKEl2al:200:201:Minnie's acct./user/home/minnie:/bin/sh
```

There are 7 fields of interest for each entry of the *passwd* file with the fields being separated by a single colon. These fields, in order, are the

- username
- encrypted password
- user ID (UID)
- group ID (GID)
- user's full name or a description of the account
- home directory for this account
- user's shell

From a security standpoint one of the most important fields is the encrypted password. On UNIX-based systems the password is not stored in its plaintext version. While this provides a certain measure of security the fact that a well known encryption scheme is used to encrypt the password makes it easier for individuals to attempt to crack them. Add to this the fact that the password file is accessible by all users on the system and it is easily seen why so much has been written about cracking UNIX passwords. There are numerous password cracking programs designed to attack UNIX passwords the most famous of which is probably **CRACK**. The majority of these programs do such things as encrypt common passwords and then compare them to a given password file. If a match is found between an encrypted word and one of the encrypted passwords then the plaintext password has been discovered. Incorporating good password selection practices (as described previously in this book) can greatly decrease the chance of a password being guessed. Unfortunately the reality of the situation is that a significant number of users will probably always practice poor password security.

Changing a password on a UNIX system is a simple matter. The program *passwd* is supplied to accomplish this task. When this program is invoked the system

will first prompt the user for the current password. This is done to verify that it is indeed the authorized user who is requesting the password change and not somebody else who has gained access to the account. This might occur if a user neglected to logoff before getting up from their terminal. After verifying the current password the system will prompt the user for the new password. This will not be echoed to the terminal so that others passing will not see it. When the new password has been entered the system will prompt the user to enter the new password again. This is done to ensure that no typographical errors were made. If the new password entered was the same both times then the password will be changed in the */etc/passwd* file (storing the encrypted version only, of course). If a mistake was made either time the system will inform the user of this fact and will leave the current password unchanged.

If a user ever forgets their password it is not an easy matter to recover it since the password is only stored in its encrypted form. A program such as *CRACK* could be run to attempt to guess it but if the user selected a good password then this may not work. It is actually an easy matter for a system administrator to edit the password file so that the password entry was blank. Such an entry might look something like the following:

```
mickey::200:200:Mickey's acct.:/user/home/mickey:/bin/csh
```

If no password is supplied in the */etc/passwd* file, the system will not prompt the user for a password during the login process. The user could then execute the *passwd* program and enter a new password and the forgotten password problem is solved. The fact that UNIX systems treat no password entry in the */etc/passwd* file as a sign that a password is not required for that specific account points out a potential security problem. System administrators should periodically check the */etc/passwd* file to ensure that all accounts have a password. If one doesn't appropriate actions (assign a temporary one for example) should be take. There are actually some limited cases when not having a password for an account may be appropriate. One such case is for accounts that have a very limited purpose. There may, for example, be an account designed to simply allow a person to run the *date* program. An entry for such an account might look as follows:

```
date::20000:200:Account to supply time and date:/tmp:/sbin/date
```

Note that for this entry the last field, normally reserved to specify the user's preferred shell, now contains the path to invoke the *date* program. Should an individual attempt to login with the userid of *date* the system would not prompt for a password. The system would also not allow the user to do anything as it would immediately execute the program specified in the last field of the */etc/passwd* file entry for userid *date* and would thus run the *date* program and then terminate the session. While the ability

to create such accounts on a UNIX-based system exists it should be used very sparingly if at all. If accounts such as the one described are created care must be taken to ensure that the program invoked does not allow shell escapes and should not be run with the UID of 0 if at all possible.

It at first may seem to be a poor idea to provide users the ability to read the password file. Unfortunately on UNIX systems this file must be readable by all because many programs need to consult it for information about the user identification number (UID), username, or home directory for a given user. As a result of the problems caused by allowing users access to the */etc/passwd* file, some versions of UNIX provide what is known as **shadow password files** to hide the encrypted passwords. The shadow file (*/etc/shadow* for example) contains the same information that the */etc/passwd* file previously did. This new file is protected, however, so that it can not be read by all users. The shadow file can still be accessed by those programs that need the information it contains but these programs generally run at the **root** level. On systems employing shadow password files the normal */etc/passwd* file will contain special values to indicate that a shadow file is being used.

As a final note on the discussion of UNIX passwords, many systems allow the system administrator to specify a maximum lifetime for passwords. This means that users will be required to change their passwords the first time they attempt to login after the password's lifetime has expired. It is a good idea to use this option if it exists on your system. The manual pages should be consulted to determine how this **password aging** is accomplished for your specific version of UNIX.

B.5 The UNIX File System

The UNIX file system consists of a hierarchical tree structure constructed of directories and files. The “topmost” level (or bottom, depending on how you want to view the tree) is appropriately referred to as **root** denoted by ‘/’. This level contains a number of directories such as */bin*, */etc*, */dev*, */tmp*, and */usr*, each of which in turn contains additional files and directories.

Associated with each file or directory are a number of attributes. Most important from a security standpoint are the file permissions and the owner. Each file or directory have two identification numbers, a user ID (UID) and a group ID (GID), associated with it. Each is owned by a single user identified by the user's unique UID. UNIX permits groups of users to be associated with various GIDs which provides for an easy method to share files. Both of these identification numbers play an important role in the UNIX access control mechanism.

UNIX employs a permission bit system to enforce access controls. Each file or directory has several sets of bits associated with it. One set of three bits describes the

access permissions granted to the user. Another set represents the permissions the user grants to others who are members of the same group. The third set specifies what all others can do to/with the file. The different types of access that can be granted are read, write, and execute. One bit in each set of three is used to indicate whether permission has been granted for each specific mode of access. For example, if a user wanted to provide total access for him/herself, allow those in the same group to read and execute the file, and all others only execute permission, the bits would be set as follows:

<u>User</u>	<u>Group</u>	<u>World</u>
rwX	rwX	rxX
111	101	001
7	5	1

Thus the access mode for this file would appear as '751'. The command used to change file permissions (to change the mode of access) is **chmod**. To set the file *payroll* to have the access modes above the command would be:

```
chmod 751 payroll
```

Just as the mode of access can be changed using *chmod*, the owner and group associated with a file or directory can also be changed. The commands to accomplish these modifications to the file or directory attributes are **chown** and **chgrp** respectively. The command used to view listings of a directory in order to find out what the directory contains is **ls**. Using the '-l' option will provide a 'long' list of the directory which will include a listing of the file permissions. An example of this command might look like the following:

```
ls -l
drwxrwxrwx 1 mickey finance 512 Jan 6 8:14 data.file
-rwxr--r-- 1 mickey finance 28035 Jan 5 10:31 info.dat
-rwxr-xr-- 1 mickey finance 234011 Nov 8 11:01 payroll
-r-x----- 1 mickey users 10532 Jan 8 9:15 vhcl.reg
```

The 'd' in front of the permissions for the **data.file** entry in the example above specifies that this is a directory. There are a couple more bits that are related to the permission bits that are of a concern to us from a security standpoint. These other bits are the SUID (set user ID) and SGID (set group ID) bits. In order to understand the function of these two bits you must first understand how a process (a running program) determines if it can have access to a specific file or not. Associated with each process are several identification numbers. Two of these are the **real** UID/GID and another two are the **effective** UID/GID. When a file (program) is executed, the real UID/GID are the UID/GID for the individual who is executing the program. Normally, if the SUID bit is not set, the effective UID/GID will be the same. Whenever a program attempts

to access another file, the system compares the file's permission bits with the UID/GID contained in the effective UID/GID for the program. If the effective UID/GID have the appropriate permissions then access is granted, otherwise it will be denied. If the SUID bit is set, it tells the system to use the owner of the file's UID as the effective UID instead of the individual attempting to run it. The SGID works the same but for the GID. Why this is important to do can best be explained using an example.

The */etc/passwd* file should not be generally modifiable for all users. The permissions for it should thus be set to not allow users to modify it (a mode of 644). The reason for this is obvious since we don't want every user to have the ability to add new accounts, change other people's passwords, or give themselves superuser privileges. There are times, however, that users have a legitimate reason to modify the file such as when they want to change their own password. We thus need to provide a limited capability to modify this file. This can be accomplished by setting the *passwd* program file's SUID bit. The program itself will be owned by *root* but will have its permissions set so that anybody can read or run it (a mode of 555 for example). When a user executes this program the real UID/GID will be that of the user who is running the program. The effective UID/GID, however, will be that of the owner of the program – in this case *root*. The program will ask for the new password for the user and will need to access the */etc/passwd* file to store it. The system will check the permission bits for */etc/passwd* and compare them with the effective UID/GID for the program. The only individual who should be able to modify the file is *root*, but since the effective UID/GID is *root*, the modification is allowed.

The SUID and SGID bits can be set in a manner similar to changing a file's other permissions bits. The SUID and SGID bits, along with another bit known as the 'sticky' bit which is not in use much today for files¹, form another set of three bits with SUID being the high order bit and the sticky bit the low order bit. The *chmod* function is used to set these bits in the same manner as the normal permission bits with the extra digit tacked on to the beginning of the mode. Setting the payroll file to have it's SUID bit set and to allow the user unrestricted access and all others to have only read permissions would be determined as follows:

¹ While not in use for files, newer versions of UNIX use the 'sticky bit' for a different purpose from what it was originally intended. On these systems the sticky bit for directories is set to keep users who aren't the owner of a file from deleting it. Without this feature, users could delete a file if they had write permissions even if they didn't own the file. Having the 'sticky bit' perform in this manner for directories is especially useful for files such as */tmp* where multiple users will have files but only the owner should have the ability to rename or delete them.

<u>Special</u>	<u>User</u>	<u>Group</u>	<u>World</u>
ugs	Rwx	rwX	rwX
100	111	100	100
4	7	4	4

The actual command to change the bits would be:

```
chmod 4744 payroll
```

The *ls -l* command can be used to display whether these extra bits have been set. Continuing with our example from above, we would see the following displayed:

```
ls -l
drwxrwxrwx 1 mickey finance 512 Jan 6 8:14 data.file
-rwxr--r-- 1 mickey finance 28035 Jan 5 10:31 info.dat
-rwsr-xr-- 1 mickey finance 234011 Nov 8 11:01 payroll
-r-x----- 1 mickey users 10532 Jan 8 9:15 vhcl.reg
```

Notice that the *x* of the users permission bits has been replaced with an *s*. This lets you know that the SUID bit is set. If an *s* appeared in place of the group *x* bit then it indicates the SGID bit is set. If in place of the world *x* bit a *t* appeared it would indicate that the sticky bit was set. An obvious question to ask at this point is how does someone know if the *x* bit is set if the system is now writing on top of it. The answer is simple. If an *s* or *t* appears in place of the *x* bit it indicates the *x* bit is also set. If instead an *S* or *T* appears it would indicate the corresponding *x* bit is not set. Thus in the above example since an *s* appears it means that the SUID bit is set and that the owner has execute permissions on the file (that the *x* bit is set).

While the SUID feature of UNIX takes care of the problem of providing only limited access to special files, it also has created a whole new level of intruder possibilities. Any file that is owned by *root* and has its SUID bit set is a potential security hole. This is especially true if the file is a shell script. The reason for the concern is that, if an intruder can find a way to exit to the shell, the entire system is wide open. At this point the intruder, who is now executing as *root* since the effective UID for the process is that of the owner, can execute any command and access any file. If an SUID program must be written, careful thought should be given as to who the owner should be. Only make *root* SUID programs when it is absolutely necessary and **never** create *root* SUID scripts. If you have any question about whether a program should be *root* SUID, don't do it.

B.6 Startup Files

As was previously mentioned, after a user logs into the system, UNIX will consult one of several startup files to set options for the user. Having these files is not a security problem in itself as long as the files are protected correctly. Only the user should have write access to the files. If an arbitrary user has the ability to write to another's startup file, instructions could be inserted that did things like copy files to another directory, change permissions on files in a certain directory, or even make a copy of the shell program with the SUID bit set and place it in the */tmp* directory.

Of particular importance in the startup files is the ability to set the default file permissions for newly created files. This can be accomplished by setting the *umask* value and is usually done so in the *.profile* or *.cshrc* files. The *umask* value works in a manner similar to a file's normal permission bits – except in reverse. The *umask* specifies the permissions that you do not want instead of the access permissions you want to grant. Therefore, if you wanted to have all files you create to have a default setting allowing you total access, others in your group the ability to only read files you create, and everybody else to have no access permissions (mode 740), you would specify its inverse (037). This will appear in the startup as:

```
umask 037
```

This line should be one of the first in the startup file as any actions taken prior to it will use the system's default *umask* instead of the one you wanted. Settings for your *umask* of either 077 or 033 should be strongly considered. As always, you can change a specific file permission at a later time by using the *chmod* command.

Several other commonly used programs have their own startup files and these too should be protected. The *ex*, *vi*, and *emacs* editors all obtain user preferences from special files. The *ex* and *vi* editors use the *.exrc* file for initialization. The *emacs* editor uses the *.emacs* file. Other related files which also must be monitored and protected appropriately include the *.forward* and *.procmailrc* files used in many email packages.

B.7 UNIX and Network Security

Most of the problems relating to security on a UNIX system can really be traced to the fact that users want to connect their system to a network. Connecting a machine to a network allows a whole new set of individuals the opportunity to attempt to access the system. Add to this the very reasonable desire to facilitate access by authorized users to systems connected to the network and we end up creating an environment

where unauthorized users can thrive. Take for example the seemingly reasonable desire for authorized users to not have to reenter their password every time they log onto a new machine on the same network. UNIX provides a facility whereby an authorized user, once authenticated on one system, can access other systems without having to reauthenticate. This can be done in one of two ways, both very simple and both very dangerous from a security standpoint.

The first method is the */etc/hosts.equiv* file. This file contains a list of trusted hosts for a specific system. Created and maintained by the system administrator, the idea behind this file is to allow groups of administrators to set up a trusted environment which can ease the burden on users. By including the name of a host in the */etc/hosts.equiv* file, a user on another host can either rlogin or execute commands via rsh on the local host without having to go through an authentication process (this assumes that an equivalent account exists for the user on both the originating host and the current host being accessed). The obvious problem with this concept of trusted hosts is that should an unauthorized individual gain access to a single host, all other systems which trust that host immediately become vulnerable to the intruder. While it can reasonably be argued that sufficient care can be exercised to limit the danger to a network if only local hosts are included in the */etc/hosts.equiv* file, including external systems is almost always a poor security practice.

A second method used to create an environment of trusted hosts is through the *.rhosts* file. A useful and highly flexible idea from a user's standpoint, this file is even more dangerous than the *hosts.equiv* file. The reason for this is that each user is allowed to create their own *.rhosts* file while the *hosts.equiv* file is restricted to the system administrator. Unrestricted use of this file by users can result in an environment where numerous hosts can be compromised.

So what can be done to limit the possibility of a problem as a result of the use of either a *hosts.equiv* or *.rhosts* file? The first step an administrator should take is to check the *hosts.equiv* file and ensure that only local systems are being trusted. Secondly, the file should be checked frequently to ensure that no + has been added (a sure sign of nefarious activity). This simple character included in the file indicates that ALL hosts are to be considered trusted hosts. This obviously presents a major security hole. The third step an administrator should take is to frequently review user's *.rhosts* files or, as is the case at numerous sites, to simply not allow them to be used. Finally, if *.rhosts* files are allowed, accounts which have an * in the password field in the */etc/passwd* file should not have a *.rhosts* file since this will bypass the normal check for the password.²

Another important file from a network perspective is the */etc/inetd.conf* file. This file is used to specify which ports the *inetd* program should listen to and which

² Another trusted host file that needs to be checked is the *.netrc* file which serves a similar purpose as the *.rhosts* file except for *ftp* instead of *rlogin* and *rsh*.

programs to start when a connection is requested. A typical *inetd.conf* file will look like the following:

```
ftp    stream  tcp    nowait  root    /usr/sbin/tcpd  wu.ftpd
telnet stream  tcp    nowait  root    /usr/sbin/tcpd  in.telnetd
finger stream  tcp    nowait  nobody  /usr/sbin/tcpd  in.fingerd -w
daytime stream  tcp    nowait  root    internal
daytime dgram  udp    wait   root    internal
chargen stream  tcp    nowait  root    internal
chargen dgram  udp    wait   root    internal
time    stream  tcp    nowait  root    internal
time    dgram  udp    wait   root    internal
```

The fields for each entry include the:

- service name
- socket type (stream, datagram raw, reliably delivered message, or sequenced packet socket)
- protocol (tcp or udp)
- wait/nowait (if datagram and timeout possible wait otherwise nowait)
- user (the userid that should run the service)
- server program (the path name for the actual program to run)
- server program arguments (any arguments to the program)

This file is important from a security perspective because it can be used to eliminate services which the administrator doesn't wish to run by simply commenting them out of the file. Some common services to eliminate include *finger* and *systat* (which can be used to obtain information about a system) as well as *tftp* (which permits file transfers without having to go through an authentication process). A second way it can be used is to make certain services more secure by specifying a special program which 'wraps' around the real service providing an extra level of security. This file should be periodically checked for modifications because intruders may try to substitute other programs for infrequently used services to insert a backdoor into the system.

While *tftp* (the Trivial File Transfer Protocol) is dangerous to run because of its lack of security features, a very common and useful service is *ftp*. Often this service is set up to allow anonymous access to a system in order to facilitate the uploading and downloading of files by individuals who wouldn't normally have access to the system. To use *ftp* anonymously a user simply enters 'anonymous' as the userid. There is no specific password that is required for this account though many systems today request the user to include their email address in order to keep a record of who has accessed the system. Setting up *ftp* on a system requires a *ftp* account with a corresponding *ftp* home directory. All files to be uploaded or downloaded will be in this home directory. When accessed, the anonymous *ftp* service executes the *chroot* function to change the

root directory in order to restrict access to the rest of the file system. Two sub-directories need to be created for this service. The first is a *bin* directory which will hold copies of the functions that can be executed while using *ftp* (such as *ls*). The other is the *etc* directory which will hold a copy of the password file which is to be used by *ftp*. The encrypted password entries for all user accounts should be replaced by an '*' or the accounts should be eliminated entirely. Leaving the accounts allows the *ls* command to list the username for the owner of a file. At the same time leaving the names of the accounts in the file allows potential intruders to learn something about what accounts exist on the system. Often a *pub* directory is also created in which all documents available for transfer are stored.

Two important services that are widely used in UNIX-based networks is the Network Information System (NIS or the updated NIS+) and the Network File System (NFS). A common desire in networks is to share information among systems to eliminate wasteful duplication. NIS allows computers to share such things as password files (*/etc/passwd*), group files (*/etc/group*), and host tables (*/etc/hosts*). NIS was designed for a small, friendly user environment and therefor has fairly poor security. Use of a firewall to limit access to NIS information from outside of a network is generally recommended. In a manner similar to NIS, NFS allows the sharing of information between systems. NFS is broader in scope in that it is designed to allow a general sharing of files and programs. Like NIS, NFS was designed for a friendlier environment and therefor also has security flaws. In particular, all information transmitted by NFS is done so in the clear and is thus subject to eavesdropping. Correctly configuring a network and its individual hosts to securely use NFS and NIS is not a trivial task and it is recommended that administrators obtain a good reference on these services (such as the O'Reilly series text covering them) before attempting to implement them.

B.8 Sources of Help

Securing a Unix system is not an easy task, especially when new security problems are constantly being discovered. Fortunately system administrators are not alone as there are numerous checklists available to help in securing a Unix-based system and a variety of organizations that publish information about security holes and patches for them. Most notable among these organizations is the Computer Emergency Response Team/Coordination Center (CERT/CC) located at the Software Engineering Institute at Carnegie-Mellon University. This organization serves as a focal point for the reporting of security problems and regularly publishes information about newly discovered problems and methods to patch or protect a system from them. A similar organization, charged with the protection of Department of Energy computer

systems and networks, is based out of the Lawrence Livermore National Laboratory. This organization, called the Computer Incident Advisory Capability (CIAC) also publishes security alerts as well as methods to secure against newly discovered security flaws. Other organizations also publish information about security flaws such as 8lgm and 10pht. Information about all of these can be easily located on the Internet.

B.9 References

- B.1 Arnold, N. Derek, UNIX Security: A Practical Tutorial, McGraw-Hill, Inc., New York, New York, 1993.
- B.2 Bach, Maurice J., The Design of the UNIX Operating System, Prentice Hall, Englewood Cliffs, New Jersey, 1986.
- B.3 Braun, Christoph, UNIX System Security Essentials, Addison-Wesley, Reading, Massachusetts, 1995.
- B.4 Curry, David A., UNIX System Security: A Guide for Users and System Administrators, Addison-Wesley, Reading, Massachusetts, 1992.
- B.5 Ferbrache, David and Shearer, Gavin, UNIX Installation Security and Integrity, Prentice hall, Englewood Cliffs, New Jersey, 1993.
- B.6 Garfinkel, Simson and Spafford, Gene, Practical UNIX & Internet Security, 2^{ed}, O'Reilly & Associates, Inc., Cambridge, Massachusetts, 1996.