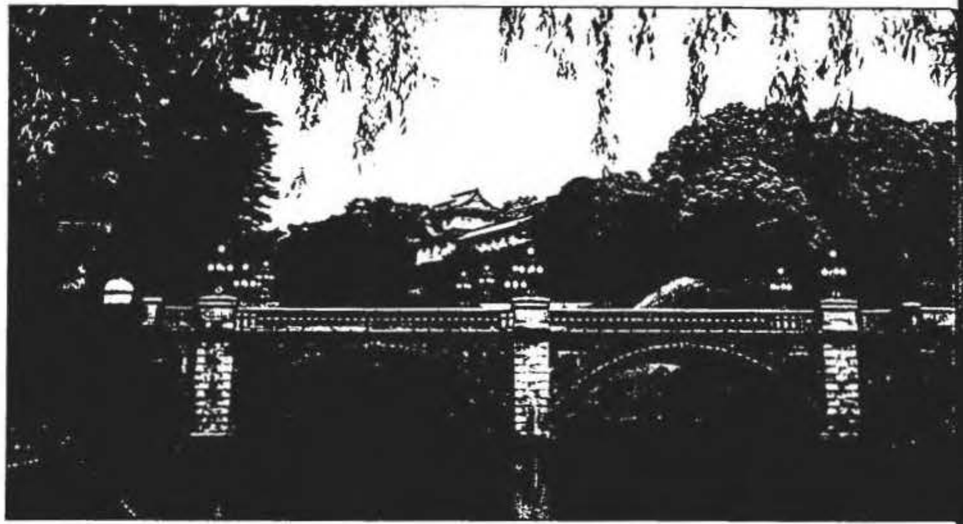
New Riders



TECHNOLOGY SERIES

INTRUSION DETECTION

Rebecca Gurley Bace

TECHNOLOGY
SERIES

INTRUSION DETECTION

What we're hearing from reviewers about Intrusion Detection. . .

"People have been working on computer intrusion detection systems for nearly 20 years. As a researcher, I am bothered that other scientists aren't familiar with the good work that has already been done, and as a consumer I am disconcerted that I don't have better commercial products to defend my systems.

Becky Bace has been there, done that, read about it, thought about it a lot, and now written it all down. Everyone who works in intrusion detection can gain something by reading this book. You can too."

Eugene H. Spafford, Professor and Director of the Purdue University CERIAS

"This book serves as a fantastic reference for the history of commercial and research intrusion detection tools. Even for practitioners of intrusion detection, this book can be an eye-opener.

Becky's book grounds the intrusion detection discussion in a way that is readable, informative, and practical."

Gene Kim, Chief Technology Officer, Tripwire Security Systems, Inc.

"I cannot imagine a consulting expert in this field who will want to be without a copy of Becky's book. Corporate managers, directors, and legal counsel need to digest these arguments as well."

Fred Chris Smith, Attorney, Santa Fe, New Mexico

"There is plenty here to point the needful System Administrator in the direction of an intrusion detection system appropriate for his current envisioned needs. But this book does much more: It provides solid perspective in a field where empty claims often dominate, and it will provide insights needed to cope with situations where existing products fall short or fail altogether to protect a system.

I am certain that this book will become an indnstry standard in intrusion detection as a discipline."

Marvin Schaefer, Chief Scientist, Vice-President, Arca Systems

"This book bridges a critical gap in the reference market. It encompasses both the principles of intrusion detection and a wealth of specific examples, enabling the reader to form a sound basis for understanding and evaluating what is happening in the field.

This book demystifies intrusion detection without oversimplifying the problem."

Ruth Nelson, President, Information System Security

The **Niju-bashi Bridge** was built in 1888 as the main access to the Imperial Palace in Tokyo. The famous "Double Bridge (Niju bashi)" is a popular name and should formally be called the "Stone Bridge." The real Double Bridge is a steel bridge that stands behind the Stone Bridge and was built at the same time. These bridges were built to replace the previous wooden bridges, giving the palace new Western-style embellishment.

INTRUSION DETECTION

Rebecca Gurley Bace

MACMILLAN TECHNICAL PUBLISHING U·S·A

Intrusion Detection

Rebecca Gurley Bace

Published by:
Macmillan Technical Publishing
201 West 103rd Street
Indianapolis, IN 46290 USA

International Standard Book Number: 1-57870-185-6

Library of Congress Catalog Card Number: 99-63273

03 02 01 00 7 6 5 4 3 2

Interpretation of the printing code: The rightmost double-digit number is the year of the book's printing; the rightmost single-digit number is the number of the book's printing. For example, the printing code 00-1 shows that the first printing of the book occurred in 2000.

Composed in Galliard and MCPdigital by Macmillan Technical Publishing

Printed in the United States of America

Trademark Acknowledgments

Warning and Disclaimer

Feedback Information

At Macmillan Technical Publishing, our goal is to create in-depth technical books of the highest quality and value. Each book is crafted with care and precision, undergoing rigorous development that involves the unique expertise of members from the professional technical community.

Readers' feedback is a natural continuation of this process. If you have any comments regarding how we could improve the quality of this book, or otherwise alter it to better suit your needs, you can contact us at `networktech@mcp.com`. Please make sure to include the book title and ISBN in your message.

We greatly appreciate your assistance.

Publisher
David Dwyer

Executive Editor
Linda Ratts Engelman

Managing Editor
Gina Brown

Product Marketing Manager
Stephanie Layton

Acquisitions Editor
Karen Wachs

Development Editor
Katherine Pendergast

Project Editor
Alissa Cayton

Copy Editor
June Waldman

Indexer
Larry Sweazy

Acquisitions Coordinator
Jennifer Garrett

Manufacturing Coordinator
Chris Moos

Book Designer
Louisa Kluczink

Cover Designer
Aren Howell

Compositors
Scan Communications Group, Inc.
Amy Parker

About the Author

Rebecca Gurley Bace is the president of Infidel, Inc., a consulting practice specializing in intrusion detection and network security technology and strategy.

Prior to founding Infidel, Ms. Bace spent 13 years in government, the first 12 as an employee of the National Security Agency (NSA). She led the Computer Misuse and Anomaly Detection (CMAD) Research program from 1989 through 1995, as a charter member of NSA's Office of Information Security (Infosec) Research and Technology (R2).

As the leader of CMAD research, Ms. Bace was responsible for championing much of the early research in intrusion detection, funding academic research at Purdue University (COAST project); University of California, Davis, (Security Lab); University of New Mexico; and Tulane University. She also served as the government's technical monitor for the Wisdom and Sense and STAR anomaly detection research projects at Los Alamos National Laboratory.

Ms. Bace's research collaborations with Dr. David Icove of the Federal Bureau of Investigation led to the commercial publication of a manual for computer crime investigation and a government study of convicted hackers. She and the CMAD workshop series she founded and sponsored were involved in the 1995 detection, traceback, and apprehension of Kevin Mitnick, at the time the FBI's most wanted computer criminal. She receives mention in Tsutomu Shimomura's book on the subject, *Takedown* (Hyperion Press, 1995). Ms. Bace received the NSA's Distinguished Leadership Award in 1995, in recognition of her work building the national CMAD community.

After leaving the NSA in 1996, Ms. Bace served as deputy security officer for the Computing, Information, and Communications Division of the Los Alamos National Laboratory. In this role, Ms. Bace was charged with determining protection strategies that allowed the Laboratory to balance needs for security with needs for availability and performance.

A native of Leeds, Alabama, Ms. Bace holds a bachelor of science degree from the University of the State of New York and a master of engineering science degree from Loyola College.

About the Technical Reviewers

These reviewers contributed their considerable practical, hands-on expertise to the entire development process for *Intrusion Detection*. As the book was being written, these folks reviewed all the material for technical content, organization, and flow. Their feedback was critical to ensuring that *Intrusion Detection* fits the reader's need for the highest quality technical information.

David Neilan has been working in the computer/network industry for more than eight years, the last five of which have been primarily devoted to network and Internet security. From 1991 to 1995, he worked at Intergraph, dealing with graphics systems and networking. David then spent four years working with DEC firewalls and network security at Digital Equipment. Since 1998, David has been working with Present Online Business Systems, LAN/WAN, and Internet security where he is designing network infrastructures to support secure LAN/WAN connectivity for various companies utilizing the Internet to create secure virtual private networks.

Robin Roberts has been in the information security industry for more than 10 years. Since 1997 she has been employed by BTG Inc., a technology integrator and services provider. At BTG she serves as an information security subject matter expert and manages an information and network security services group with particular focus on customers from the intelligence community. From 1986 to 1997, Robin worked for the Central Intelligence Agency, managing the Information Security R&D Program and providing subject matter expertise to a variety of agency projects.

Stephen E. Smaha was founder and CEO of Haystack Labs, Inc., which designed, implemented, and fielded software-based intrusion and misuse detection systems starting in 1989. Before launching their first commercial product in 1993, Haystack Labs did research and development work on intrusion detection systems for a variety of government agencies and their contractors, including the FBI, National Security Agency, Department of Energy, the U.S. Air Force, and some unmentionables. Haystack Labs, Inc., was acquired in October 1997 by Trusted Information Systems (TIS). At TIS, Smaha served as vice president for technology until that company's acquisition by Network Associates in April 1998. Since that time, he has served on several computer company boards of directors and technical advisory boards and is actively involved in mentoring startup companies. Prior to founding Haystack Labs, Smaha developed computer security systems for military customers at Tracor Applied Sciences, managed an artificial intelligence software group at Schlumberger, designed office automation workstations at Syntrex Corp., and wrote biostatistics software for Health Products Research. Smaha is a well-known speaker and contributor to Interop, COMDEX, Internet World, and a variety of security-related forums. He has served on federal and state-level expert panels on security and privacy. Smaha's undergraduate degree is from Princeton University in math and philosophy. He has a master's degree from the University of Pittsburgh in philosophy and a master's degree from Rutgers University in computer science.

Fred Chris Smith practices law in Santa Fe, New Mexico, where he has lived since 1978. Since 1985 he has also consulted from time to time with the Los Alamos National Laboratory about various digital evidence analysis tools and other computer forensic technologies developed by the national labs. He currently consults with the lab in an ongoing effort to make new computer forensic tools and techniques available to public law enforcement and to private computer security professionals. He served as the director of special prosecutions and investigations for four consecutive New Mexico attorneys general. Since 1989 he has worked with SEARCH and recently helped to develop the advanced Internet investigation course curriculum for state and local law enforcement officers, which he helps to teach in Sacramento, California. He currently serves on the National White Collar Crime Center Executive Director's Advisory Board in Richmond, Virginia. Over the past 10 years, Fred has developed training programs and spoken to numerous state and federal agencies about computer crime and new developments in theories of legal liability resulting from an increased use of networked software applications in commerce. He works as a consultant for groups and companies from the private sector on investigation and litigation strategies where electronic evidence is involved. His most recent publication is a manual for the National Coalition for the Prevention of Economic Crime, *Forming Partnerships for the Prosecution of Computer Network Intrusions,* which will be published sometime after Y2K. Fred attended the University of Michigan as an undergraduate and received his law degree from Stanford in 1972.

Christopher Wee has been a researcher in intrusion detection and network security since 1991. His research interests are in host-based audit monitoring, the exploitation of vulnerabilities in network protocols, and the specification of security policies. As a graduate student and postdoctoral researcher at University of California, Davis, he worked on the DIDS, LAFS, GrIDS, and IDIP intrusion detection systems. Chris is currently a senior Infosec analyst with Intel Online Services, Inc. He holds a bachelor of science degree in electrical engineering and a master's degree and doctorate in computer science from University of California, Davis.

Dedication

To the "Graybeards" and "Nobeards" of computer security—may we someday get it right.

In loving memory of Joey Bace,

(1985–1994)

who taught his mom what matters most.

Acknowledgments

During the writing of this book, as in the rest of my life, I've been blessed with an abundance of extraordinary people who have spun a web of support around me.

I am deeply indebted to Steve Smaha, who has been my intrusion detection muse for many years. He, Jessica, and Rebecca have been a source of support and inspiration to me through the past decade. It was at Steve's behest that I tackled writing this book, and he was the source of much entertaining and informative discussion throughout the process.

Jennifer Garrett, Katie Pendergast, Alissa Cayton, and Linda Engelman of Macmillan Technical Publishing have been a joy to work with, encouraging and guiding me through the totally alien landscape of the publishing business.

My colleagues in network and information security make up a wise, intelligent, and incredibly entertaining community. They have been generous with information and encouragement, responding to my requests for opinions and explanations with unfailing good humor, funny email, fresh gossip, and profound insight. Special thanks go to Jim Anderson, Dorothy Denning, Gene Spafford, Bob Abbott, Marv Schaefer, Ruth Nelson, Marcus Ranum, Kevin Ziese, Adam Shostack, Chris Wee, Fred Smith, Drew Gross, Carolyn Turbyfill, Robin Roberts, Stephanie Fohn, Gene Kim, Ron Gula, and Dave Icove.

My former colleagues in the National Security Agency are brilliant and dedicated professionals who perform a critical, though all too often thankless function in our society. I consider it an honor to have been part of that organization, and I salute them for their support of the nation.

Finally, my family has been a source of immense joy and enlightenment to me. This includes the family to which I was born as well as the family that has gathered around me in the form of close and steadfast friends. I'm fortunate to have so many who have opened their hearts and lives to me. I am especially indebted to Terri Gilbert and to Paul Bace for their love, support, and patience as I wrote this book.

OVERVIEW

CONTENTS

INTRODUCTION

Computer and networking technologies dominate much of our lives today. Many of us rely on these technologies in our everyday lives: Our work, leisure, community, transportation, and communications are enabled by these systems. The widespread panic associated with Y2K problems and the subsequent threat to public infrastructures demonstrates how dependent we are on support structures ultimately controlled by computers.

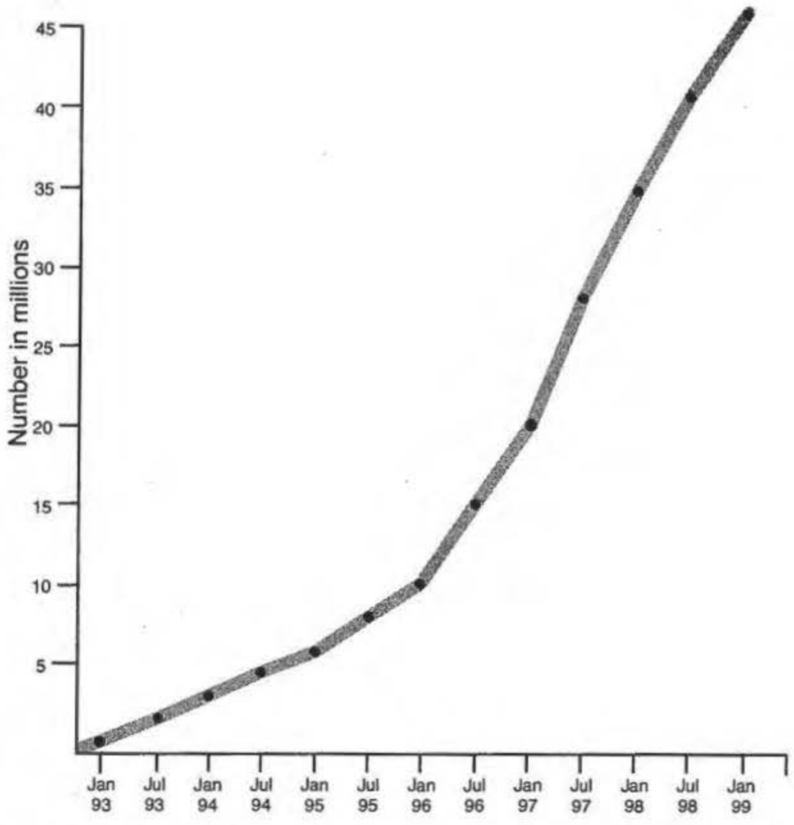
Even as we rely on these systems, we're painfully aware of the flaws and imperfections in them. System failures are blamed for catastrophes ranging from airliner crashes to medical equipment failures. Media coverage of hacker incidents and disastrous system failures capture our attention and elevate public concern about the dependence upon these all too fallible machines. However, this concern is tempered by the appeal of new technologies, which offer near-magical capabilities to us. Even commerce has been transformed by the vision of a society that purchases goods and services in a virtual marketplace, where storefronts are built of bytes and network packets, not brick and mortar.

Over time, the wide-scale adoption of new consumer technologies follows a rather predictable pattern. First the technology is introduced, and early adopters of the technology utilize it, becoming the leading edge for the rest of the populace. Depending on the nature of the technology as well as the circumstances of the society, this phase is followed by mainstream adoption of the technology. As access to the new technology increases, some users exploit its capabilities to drive progress. Unfortunately, others utilize the technology to cause injury and to facilitate criminal activity. Ultimately, in response to public concern, the legal and law enforcement communities institute statutory and enforcement measures to deal with these problems.

Our experience with computers and networks have been no different. Initially, the lack of access to computers and the high cost of building and maintaining them limited the problems associated with security breaches. As the amount of critical information entrusted to the systems increased along with the remote access capabilities, the security problems became apparent.

The advent and rapid growth of the ARPANET, born of a partnership between government and academia, served to accelerate this security exposure. ARPANET was designed to function in a small community in which members were trusted and information had little perceived value. "Handshake" agreements were the order of the day, the number of account holders was small, and many users of the network knew each other.

From the figure below, which shows the growth of the Internet over the last few years, it is apparent that those days of the early Internet are long gone. Gone with them is the high-trust culture that defined the network community of that era. Many business organizations use the Internet as a setting for their most critical business operations. Government organizations use the Internet as a means of providing public access to public records and information, with future plans to utilize the network as a staging ground for elections and referenda.



Internet Domain Survey, January 1999, Number of Hosts Advertised in the DNS[1]

In this network world, the needs for security and appropriate systems of control are apparent. The marching orders for those who would secure computer systems and networks are ambitious indeed. The security achieved must be reasonable yet sufficient, balancing needs for accountability with equally important needs for privacy. It must be flexible enough to accommodate a global range of statutes and regulations yet consistent enough to allow tracking of criminals across multiple jurisdictions.

The blend of management and technical measures necessary to meet these security requirements is rich and complex, a veritable smorgasbord of topics and issues for the interested researcher or developer. The area of security audit and intrusion detection has become an important part of computer and network security. The functions provided by these technologies serve the goals of security both directly, by providing traceback and detection capabilities, and indirectly, by monitoring the health and trustworthiness of other security mechanisms in the system.

Defining Intrusion Detection

Intrusion detection is the process of monitoring the events occurring in a computer system or network, analyzing them for signs of security problems. You can probably think of analogous monitoring systems in other areas, including burglar alarms and video-monitoring systems found in convenience stores and banks. On a grander scale, civil defense and military early-warning systems fall into this functional category. Although the monitoring strategies and targets differ, the general idea remains the same—these systems all provide sentinel functions, alarming and alerting responsible parties when activities of interest occur.

Note

The term intrusion detection is also used by the military to refer to systems that monitor physical entities (such as communications cables) for evidence of tampering or other physical alteration. Military standards describe system functions and benchmarks for this area. In this book *intrusion detection* refers to the monitoring, detection, and response functions that target activity in computer systems and networks.

Intrusion detection is a relatively young technology, as is noncryptologic computer security in general. The bulk of intrusion detection research and development has occurred since 1980. However, this research has produced a wide range of proposed solution strategies for accomplishing intrusion detection goals.

The youth (and subsequent immaturity) of intrusion detection brings other complications to bear. A gulf still exists between theoretical and practical aspects of intrusion detection. This situation creates all sorts of temptations for those researching and developing products in the area. For example, there are temptations to define terms on the fly and to develop proprietary solutions that aren't interoperable with other parts of the system security or management infrastructure. Another strong temptation is to claim that a favorite solution or approach solves all problems regardless of the validity of the claim.

These issues will eventually be resolved, driven by customer need and increased funding for research and development. The importance of intrusion detection in defensive information warfare is apparent, and the government has announced plans for additional expenditures in this area.

By Way of Introduction

I've spent the last 10 years immersed in the world of intrusion detection. In that time, I've seen a plethora of problem statements, proposed solution strategies, commercial products, and experts in the area come and go. I have directed government research for the area, attempting to marry research interests with operational needs. I've performed research of my own, exploring the techniques utilized by hackers to defeat security mechanisms. After years in the research community, I moved to a security management job and utilized some of the products in a challenging operational environment. And, coming full circle, I've devoted the last couple of years to working with security novices as they utilize existing commercial products and build new ones. Throughout this time, I've been delighted to see the successful transfer of many intrusion detection products to the commercial marketplace.

That delight, however, has been tempered with frustration. I see a commercial product market that utilizes but a fraction of the insight the last 15 years of research have produced. I see practitioners and developers who prescribe and build systems without ever asking end users what they really need and how best to integrate intrusion detection capabilities with their existing systems and practices. I see research and development initiatives that demonstrate no apparent understanding of the problems users face or of the work that has been already done. Finally, I see continuing resistance to the free and open discussion of intrusions and vulnerabilities with those who desperately need that information to protect their systems.

Despite these problems, I still believe in the value and future of intrusion detection as an integral part of computer security. Security, well done, can protect our privacy and the information that defines so much of who we are in this virtual universe. It also enables the formation of new and transformative communities by the users of the Internet, allowing us to collaborate in solving many of the pressing problems before us in the new millennium.

It is in this sprit that I wrote this book. It represents an opportunity to record the experiences of one blessed with the chance of a lifetime—to witness the growth of a technology from concept to commercial product. I hope that the information included enables you to include intrusion detection in your arsenal as you work toward achieving your system security goals.

Endnote

1. The survey data charted in the figure is provided by Network Wizard. The data is also available on the Internet at `http://www.nw.com/`.

CHAPTER 1

The History of Intrusion Detection

"Life was simple before World War II. After that, we had systems."
- Admiral Grace Hopper

When I explain intrusion detection to those not familiar with network security, it's usually easy to describe what intrusion detection systems do: "It's a burglar alarm for computers and networks," or "It looks for criminals breaking into a computer system and lets someone know about it." Most people understand that when systems handle things that are considered valuable, the systems themselves are natural targets of attack.

Although the goals of intrusion detection systems may be intuitively obvious to both technical and nontechnical users, many people are not aware of the history of intrusion detection research and development. This lack of information results in repeating mistakes made in the past or needlessly settling for suboptimal approaches to critical functions.

Intrusion detection has merged traditional electronic data processing (EDP) and security audit with optimized pattern-matching and statistical techniques. Intrusion detection has become an integral part of modern network security technology. In this section, I describe the history of audit and intrusion detection from the perspective of the people who did the initial research and development and their projects.

1.1 Audit: Setting the Stage for Intrusion Detection

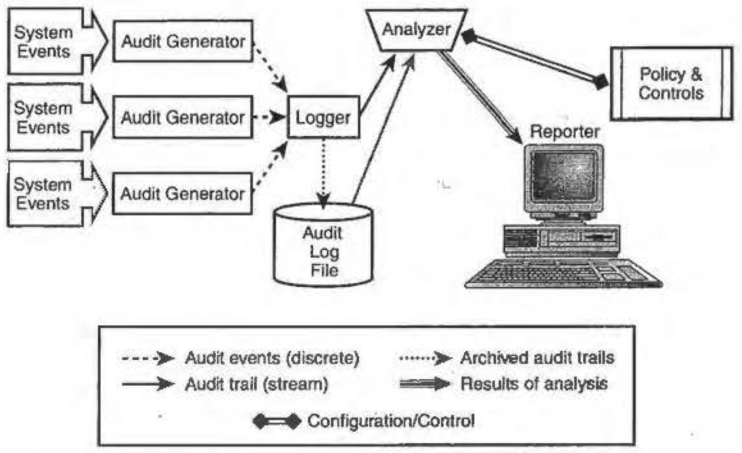
Before intrusion detection, there was audit. *Audit* is defined as the process of generating, recording, and reviewing a chronological record of system events. People audit systems to accomplish a variety of goals. These goals include the following:

- To assign and maintain personal accountability for system activities
- To reconstruct events
- To assess damage

- To monitor problem areas of the system
- To allow efficient damage recovery
- To deter improper use of the system

Figure 1.1 shows a simple diagram of an audit process. It includes the audit trail generator, logger, analyzer, and a reporting mechanism. Note that this audit process can apply to both manual as well as computer processes.

Figure 1.1 Basic Audit System



A premise that underlies all audit processes is that a set of rules governs the audit. The exact form and substance of this rule set varies, depending on the context of the audit process. In financial audit, the rule set may comprise generally accepted accounting principles, procedures, and practices. In management audit, the rule set comprises management controls, procedures, and practices meant to assure certain goals for the business. These business goals include items such as judicious use of resources, maximization of profits, minimization of costs, compliance with applicable laws and regulations, and appropriate control of risk.

In the special case of computer security audit, the rule set is usually articulated in a *security policy*. As you'll see in Chapter 3, "Information Sources," for intrusion detection to perform a complete and competent analysis of the audit data, additional rules that are not so clearly articulated in policy form are required. Note that the rules against which the system is checked affect the entire audit process. Consequently, we design the audit mechanism to collect the data elements necessary to detect noncompliance with the rules. In addition, the size and complexity of the rule set also drives decisions we make regarding the storage and analysis requirements of the audit analysis and archive systems.

1.1.1 Differences between Financial and Security Audit

Although the processes of traditional financial and management audit appear to be identical to computer security audit at the highest levels of abstraction, significant differences exist between the problems addressed. Financial audit addresses tracking transactions from cradle to grave; in other words, financial audit involves tracing the trail of evidence that links a chain of transactions to the summary figures in a financial statement. Another assumption for financial audit is that it reviews a *deterministic* process; in other words, a process that allows transactions to be traced both forward and backward in the system from any entry point within the chain of transactions that occurred within that system. The process is also assumed to be chronologically ordered in some consistent fashion.

Where these properties may still apply to accounting systems, they do not apply to the processes of many modern computer systems! Furthermore, there is no metric analogous to the summary balance in financial systems that can serve to confirm that the system is in a secure state at any given time. Finally, as the security audit is considered part of the *protection envelope* for the system it monitors, the security audit is likely to be a target for attack. Therefore, additional requirements exist for the protection of the audit mechanism, the system on which it runs, and the audit trail generated by the mechanism.

1.1.2 Audit as a Management Tool

In early computing environments, computers were large, novel, and expensive to acquire, operate, and maintain. Therefore, access to computing time was precious. Audit mechanisms in early operating systems were devoted to accounting for every microsecond of computing time (and, of course, billing users for the use of this time!).

As computers became more common and the number of business applications increased, someone noticed that the information collected in audit trails was useful for purposes other than billing. In particular, the information from audit trails could allow management to understand how the computer was being utilized, whether this use was appropriate to the organizational goals, and where resources might require modification.

Close on the heels of this discovery came the realization that these audit trails also supported the investigation of problems involving misuse of the systems. This area was of interest to organizations that used computers to conduct financial transactions. It was of even more interest to organizations that used computers to handle sensitive information. Hence came the earliest requirements for manual audit trail review. These requirements gave rise to the two major camps in computer audit. EDP audit served the objectives of the business-computing community. Security audit initially focused on needs in the military and government-computing world.

1.1.3 EDP Audits and Early Computer Security

Perhaps the first documented work defining a specialized EDP audit program was initiated in the mid-1950s. One of the earliest major corporate users of computing technology was the Bell Telephone System (which was the legally designated telephone monopoly for the United States at that time). A Bell Telephone Laboratory task force assembled in 1954 to analyze the future use of computers in the telephone business. This task force established the need for EDP auditing, differentiated it from the primarily paper-driven audit processes that marked system audit up to that point, and, by 1959, had an audit staff that was integrally involved in the design of the first large-scale computerized telephone-billing system.

EDP Objectives

The objectives of EDP audit mirror those of the manual business audit process that came before it. These objectives were to curb losses associated with error and fraud. Bell Telephone personnel noted, even in the 1950s, that to perform meaningful audits, auditors had to be able to evaluate software for the set of controls implemented by the programmers.

The concerns stated by the audit researchers were that auditors were attempting to audit "around the machine," not "through the machine." Auditing *around the machine* meant that auditors limited their audit examinations to the input and output of computer systems. Auditing *through the machine* meant that auditors had to understand what the system and its programs were doing and how they did it. Furthermore, the auditor needed to be able to use the computer itself to perform some of the audit checks, utilizing specialized audit programs to perform data sampling, seeding input data with test cases, and filtering transaction data for special cases.[1]

Many of the concerns articulated by EDP auditors are mirrored in the computer security world. Some computer security specialists have practiced within the EDP audit world for many years, with special interest communities that have functioned since the 1960s. An active software product market serves the specialized needs of EDP auditors, most of them facilitating searches and statistical checks of financial and other business process information.[2]

1.1.4 Audit and Military Models of Computer Security

The U.S. Department of Defense (DOD) backed an extensive research effort during the 1970s, which explored security policies, guidelines, and controls for operating "trusted systems," culminating in the DOD Security Initiative of 1977.

Trusted systems were defined as "systems that employ sufficient hardware and software assurance measures to allow their use for simultaneous processing of a range of sensitive or classified information."[3] Thus, trusted systems were designed from the ground up in a way that allowed military and intelligence organizations to place information of different sensitivity levels (typically corresponding to levels of classification) on the same computer system. The Trusted Systems Initiative provided a venue in which the computer security experts of the era explored the features and protections that were necessary for trusted systems to function. (Trusted systems are discussed in more depth in Chapter 3.)

During the initial explorations, researchers debated whether security audit mechanisms would contribute to the assurance level of a trusted system. Ultimately, the audit mechanism was indeed included as part of the *Trusted Computer System Evaluation Criteria*[4] ("Orange Book") requirements for systems evaluated at trust levels C2 and above. The series of documents that outlined the DOD's Trusted Systems Initiative are often referred to as the "Rainbow series," in a reference to the brightly colored covers of the documents.

A document, which addresses the issue of audit in trusted systems, is included in the Rainbow series. It is the "Tan Book," titled *A Guide to Understanding Audit in Trusted Systems.* → Audit (TCSEC, C2)

Tan Book

The Tan Book outlines five security goals for audit mechanisms:

- To allow the review of patterns of access (on a per-object and per-user basis) and the use of protection mechanisms of the system
- To allow the discovery of both insider and outsider attempts to bypass protection mechanisms
- To allow the discovery of a transition of a user from a lesser to a greater privilege level; for example, when a user moves from clerical to system administrator roles
- To serve as a deterrent to users' attempts to bypass system-protection mechanisms
- To serve as yet another form of user assurance that attempts to bypass the protection will be recorded and discovered, with sufficient information recorded to allow damage control

Although the Tan Book (as well as much of the Rainbow series) reflects a rather centralized mainframe view of computing, its principles of security audit still apply.[5]

1.2 The Birth of Intrusion Detection

As the speed, size, and number of computers increased over the 1970s, the need for computer security became increasingly apparent. The government, realizing that the traditional audit community had experience in dealing with tracking activities that took place on computers, made a decision to enlist its assistance. In 1977 and 1978, the National Bureau of Standards convened meetings of representatives of government and commercial EDP auditing organizations, which produced reports on the state of security, audit, and control at that time.

At the same time, the DOD, increasingly concerned about security issues associated with the proliferation of computer usage in military systems, increased its scrutiny of computer audit as a security mechanism. This task fell into the able hands of James P. Anderson.

1.2.1 Anderson and the Audit Reduction Problem

James P. Anderson is acknowledged as the first person to document the need for automated audit trail review to support security goals. Anderson, who published the Reference Monitor concept in a planning study for the U.S. Air Force, wrote a report in 1980 that is considered to be the seminal work on intrusion detection.[6] In this report, he proposed changes to computer audit mechanisms to provide information for use by computer security personnel when tracking problems. The goal of *audit reduction*, the elimination of redundant or irrelevant records from security audit trails, was first articulated in this report.

A major classified customer who handled sensitive data in mainframe environments featuring stringent security-management controls motivated Anderson's work. This customer had policies that required the auditing of all computer activity, supported by a security staff that manually reviewed audit trails and investigated problems uncovered in the audit trail review. The task of performing this manual review and investigation was becoming onerous as computing volume increased. Furthermore, the customer's security staff was discovering that its ability to detect some security problems in its audit review was jeopardized by missing or superfluous information in the audit trails.

Anderson proposes a taxonomy for classifying risks and threats to computer systems (see Figure 1.2) that differentiates between external and internal sources of problems on both a full system and per file/object basis. This articulation of concerns was helpful in structuring requirements for audit trail content.

Andi
E

Figure 1.2 Anderson's Threat Matrix

	Penetrator not Authorized to use Data/Program Resource	Penetrator Authorized to use Data/Program Resource
Penetrator not Authorized Use of Computer	CASE A: External Penetration	■
Penetrator Authorized Use of Computer	CASE B: Internal Penetration	CASE C: Misfeasance

Anderson's report articulates several goals for security audit mechanisms:

Anderson Goals

- They should provide enough information for security personnel to be able to localize problems, but not so much that the audit trails themselves provide enough information to enable an attack.
- To optimize audit trail content to allow detection of problems, one must be able to collect information on a variety of system resources.
- To detect insider abuse of systems, the audit analysis mechanism should be able to discern some notion of "normal" activity for a given resource (where a user is considered to be a resource).
- The design of an audit mechanism should take the strategy of a system attacker into account.

Anderson points out that when a violation occurs in which the attacker attains the highest level of privilege, such as *root* or *superuser* in UNIX, there is no reliable remedy. For this worst-case scenario, one can instrument a system with embedded audit mechanisms that monitor CPU and other systems internals, but this defense is not particularly durable.

He devotes some time to the problem associated with *masqueraders*, those adversaries who access systems using purloined user IDs and passwords. To the system, masqueraders appear to be legitimate users. Anderson suggests that some sort of statistical analysis of user behavior, capable of determining unusual patterns of system use, might represent a way of detecting masqueraders.[7] This suggestion was tested in the next milestone in intrusion detection, the IDES project.

1.2.2 Denning, Neumann, and IDES

From 1984 to 1986, Dorothy Denning and Peter Neumann researched and developed a model for a real-time intrusion detection system, named the Intrusion Detection Expert System (IDES). This research, funded by the U.S. Navy's Space and Naval Warfare Systems Command (SPAWARS), proposed a correlation between anomalous activity and misuse. *Anomalous*, as defined in this project, meant "rare or unusual" in a statistical sense (in effect, outside of some statistical characterization of normal).

This assumption served as the basis for many intrusion detection research and system prototypes of the 1980s. Denning's 1987 paper on the topic is considered to be another seminal work in intrusion detection.[8]

The IDES model is based on *profiles*, data structures that use statistical metrics and models to describe the behavior of system subjects (primarily users) with respect to system objects. *Activity rules* specify actions to be taken at a given time (either the generation of an event record or the end of a time interval). The statistical metrics and models allow the system to evaluate behaviors against both fixed and dynamic measures of normality.

Denning and Neumann's model was instantiated in the landmark IDES prototype system, developed at SRI International from 1986 to 1992.

The IDES prototype system used a hybrid architecture, comprising an *anomaly detector* and an *expert system*.

The anomaly detector used statistical techniques to characterize abnormal behavior. The expert system used a rule-based approach to detect known security violations. The expert system was included to mitigate the risk that a patient intruder might gradually change his or her behavior over a period of time to defeat the anomaly detector. This situation was possible because the anomaly detector adapted to gradual changes in behavior to minimize false alarms.

The IDES prototype system was developed on a TOPS-20 system. Principal investigators and researchers of the IDES prototype included Peter Neumann, Harold Javitz, Teresa Lunt, R. Jagganathan, and Fred Gilham.[9]

1.2.3 A Flurry of Systems through the 1980s

The Anderson report and the work on IDES launched a cluster of research prototype systems over the next few years. We will mention several of these efforts, outlining the general assumptions, approaches, architectures, and results of each. Some will be covered in greater detail as we explore details of various approaches to analysis.

1.2.3.1 Audit Analysis Project

In 1984 to 1985, a research group at Sytek conducted a project funded by the U.S. Navy's SPAWARS Command. The Automated Audit Analysis project prototyped a system that utilized data collected at the shell level of a UNIX machine running in a research environment. The data was then analyzed by using database tools. This research demonstrated the capability to distinguish normal from abnormal system usage. Principal researchers for this effort were Lawrence Halme, Teresa Lunt, and John Van Horne.[10] Lunt went to SRI International, after the completion of this project, to lead the IDES project.

1.2.3.2 Discovery

Discovery is an expert system designed for detecting and deterring problems in TRW's online credit database. It was sponsored as an internal research and development project at TRW. This system was a bit different from the monitoring environment of the other systems from this era in that the database application, not the operating system, was monitored for intrusions and misuse. The goal for Discovery was to process daily inquiry activity in search of unauthorized inquiries. The processing load for the database system monitored by Discovery was estimated at approximately 400,000 inquiries per day, representing approximately 120,000 access codes. Audit trails were collected and the system was run against the data in batch mode.

Discovery used statistical inference to locate patterns in the input data. The system was designed to detect three types of abuse scenarios: unauthorized access, insider misuse, and invalid transactions. Discovery's statistical engine was written in COBOL, with an expert system written in an AI shell. Both parts of the system ran on an IBM 3090. The principal architect was William Tener.[11]

1.2.3.3 Haystack

Haystack is a system that was developed by Tracor Applied Sciences, Inc. (initially, from 1987 to 1989) and Haystack Labs (from 1989 to 1991) for the U.S. Air Force Cryptologic Support Center. Haystack was designed to help security officers detect insider abuse of Air Force Standard Base Level Computers (SBLC). These computers, Sperry 1100/60 mainframes running early 1970s vintage operating systems, were used to do traditional mainframe data processing tasks. The tasks (accounting, finance, inventory control, and personnel) handled data considered "unclassified but sensitive." The computers were running operating system software that was equivalent to a Trusted Systems evaluated level of B1, which included extensive audit mechanisms. The systems generated audit records reflecting more than a million events per week.

Haystack was implemented on an Oracle database management system running on an IBM-AT clone. Haystack's analysis engine was written in ANSI C and SQL and performed anomaly detection in batch mode, which meant that it periodically downloaded the audit trail file from the target SBLC system and then processed it.

Haystack characterized the information from system audit trails as sets of "features." Examples of features include session duration, number of files opened, number of pages printed, number of CPU resources consumed in the session, and number of subprocesses created in the session. Because there was no notion at the time of which features were most effective in detecting intrusions, the system included more than 30 features for each session. It used a two-stage statistical analysis to detect anomalies in system activity. The first stage, which checked each session for unusual activity, checked each feature against specified bounds and then performed a statistical test to determine whether the number of features that exceeded bounds was large enough to indicate unusual behavior. The second stage used a statistical test (the Wilcoxon-Mann-Whitney Ranks test) to detect trends in sessions. The combination of the two techniques was designed to allow detection of both "out-of-bounds" activities as well as activities that gradually deviated from normal over time. Haystack was fielded in 1992 and used on all U.S. Air Force SBLC systems for several years. The principal architect of Haystack was Steve Smaha.[12]

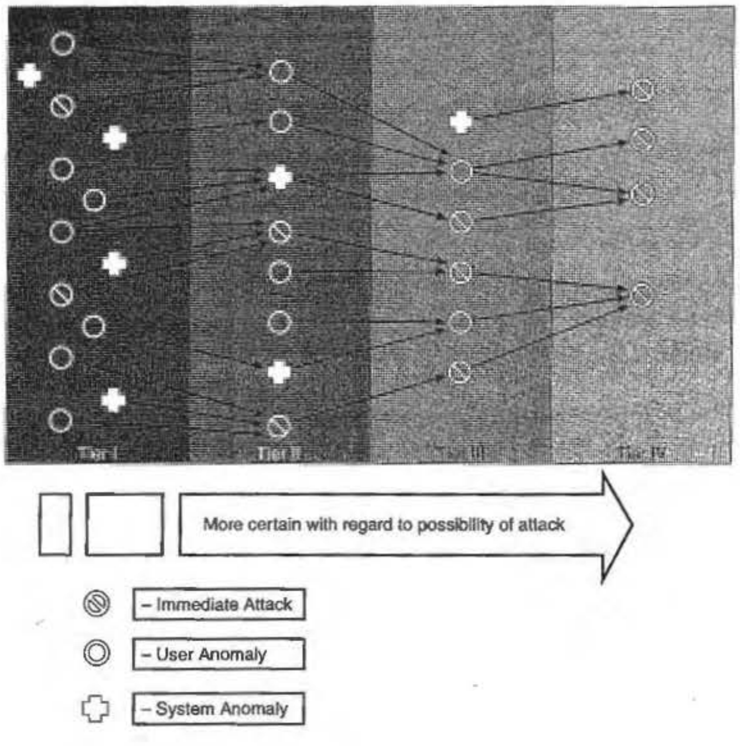
1.2.3.4 MIDAS

Multics Intrusion Detection and Alerting System (MIDAS) was developed by the National Computer Security Center (NCSC) to monitor the NCSC's Dockmaster system, a Honeywell DPS 8/70 running Multics, a highly secure operating system.

MIDAS was designed to take data from Dockmaster's answering system audit log. (On Multics, the answering system handled the user logins and password challenges, spawning user sessions.) Multics augmented the audit log data by collecting other information from the system. This data was organized, used to construct session profiles, and then compared to user profiles of normal behavior. MIDAS, like IDES and several other systems of the era, used a hybrid analysis strategy, combining statistical anomaly detection with expert system rule-based approaches.

MIDAS's expert system used a forward-chaining algorithm featuring four levels of rules. Figure 1.3 describes this heuristics structure in more detail. In addition to this rulebase, MIDAS kept user and system-wide statistical profiles in a statistical database, which was updated at the end of each user session. The statistical and rule-based analysis portion of MIDAS was coded in LISP and ran on a Symbolics workstation. MIDAS was placed online in 1989 and monitored Dockmaster through the mid-1990s.[13]

Figure 1.3 MIDAS Expert System Architecture



Note

MIDAS, one of the first intrusion detection systems that monitored an operational system connected to the Internet, gave a fascinating view of the internet threat. Dockmaster was an attractive target for attack due to its affiliation with the Defense Department. Perhaps the most interesting insight that MIDAS gave us was a demonstration of the value of strong identification and authentication (I&A) mechanisms. In the late 1980s, after MIDAS came online, the NCSC decided to utilize token-based I&A as a replacement for weaker password mechanisms. In the scheme selected, users had a calculator-style token, protected by a PIN. When someone logged into Dockmaster, the system issued a multidigit challenge; this challenge was entered into the token, and the resulting multidigit response was sent back to Dockmaster, at which time the user session began.

MIDAS allowed the Dockmaster security staff to see the effects of enacting this stronger login mechanism. Doorknob-rattling and password-guessing incidents dropped dramatically after the token-based I&A system was in place. This effect emphasizes again the importance of thinking holistically when determining a protection strategy for systems.

The intrusion detection system could have continued to record attempts to hack into the system by password-guessing or other such attacks on weaker I&A mechanisms. Attackers might have been deterred, but only after a great deal of time and energy were spent on investigating the incident—tracking down the attacker, going through the phases of a criminal investigation and prosecution, and hoping for a sufficiently harsh sentence to discourage further attacks. The desired effect (to make external attackers stop) was much more quickly and easily accomplished by using a strong protection mechanism at the system access point.

Another even more important point is that, despite strengthening I&A to a point where external attackers were thwarted, the organization continued to run MIDAS to monitor for insider abuse.

1.2.3.5 NADIR

Network Audit Director and Intrusion Reporter[14] (NADIR) was developed by the Computing Division of Los Alamos National Laboratory to monitor user activities on the Integrated Computing Network (ICN) at Los Alamos. This network is Los Alamos's main computer network and serves more than 9,000 users—connecting supercomputers, local and remote terminals, workstations, network services machines, and data communications interfaces. NADIR monitors the network by processing audit trails generated by specialized network service nodes. It was designed to run on Sun UNIX workstations and, like many other systems of the time, it performs a combination of expert rule-based analysis and statistical profiling. NADIR is written in SQL and runs on a Sybase database management system, using some of Sybase's internal triggers and other features.

NADIR remains one of the most successful and durable intrusion detection systems of the 1980s and has been extended to monitor systems beyond the ICN at Los Alamos. NADIR continues to monitor the ICN at the time of this publication, and the team continues to modify the system to accommodate new threats and target systems. The principal architect for NADIR is Kathleen Jackson.

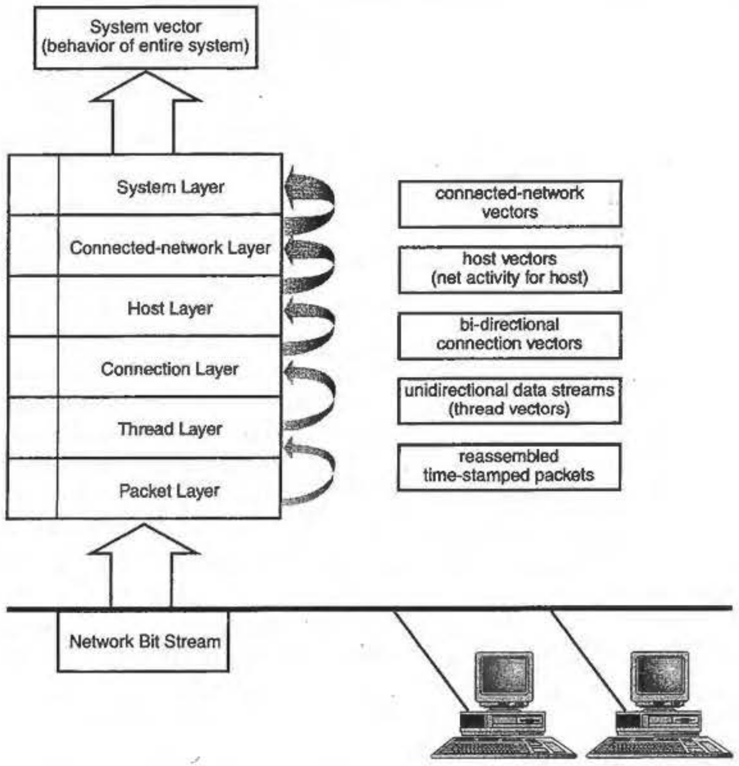
1.2.3.6 NSM

The Network System Monitor (NSM) was developed at the University of California at Davis to run on a Sun UNIX workstation. It represented the first foray into monitoring network traffic and using that traffic as the primary data source. Before this time, most intrusion detection systems consumed information from operating system audit trails or keystroke monitors. The general architecture of the NSM is still reflected in many commercial intrusion detection products at the time of this publication. The NSM functioned by doing the following:

- Placing the system's Ethernet network interface card into promiscuous mode (in which each network frame generates an interrupt, thereby allowing the monitoring system to listen to all traffic, not just those packets addressed to the system)
- Capturing network packets
- Parsing the protocol to allow extraction of pertinent features as shown in Figure 1.4
- Using a matrix-based approach to archive and analyze the features, both for statistical variances from normal behavior and for violations of pre-established rules

NSM was a significant milestone in intrusion detection research because it was the first attempt to extend intrusion detection to heterogeneous network environments. It was also one of the first intrusion detection systems to run on an operational system (the computer science department local area network at UC Davis). In a widely cited, two-month test of NSM, it monitored more than 111,000 connections on the network segment, correctly identifying more than 300 of them as intrusions. The system administrators for the network discovered less than one percent of these intrusions. This test emphasized the need for and the effectiveness of intrusion detection systems as part of the protection suite. Principal architects for NSM were Karl Levitt, Todd Heberlein, and Biswanath Mukherjee of the University of California at Davis.[15]

Figure 1.4 NSM Architecture


System vector
(behavior of entire system)
System Layer
Connected-network Layer
Host Layer
Connection Layer
Thread Layer
Packet Layer
connected-network vectors
host vectors (net activity for host)
bi-directional connection vectors
unidirectional data streams (thread vectors)
reassembled time-stamped packets
Network Bit Stream

1.2.3.7 Wisdom and Sense

Wisdom and Sense[16] was an anomaly detection system developed by the Safeguards and Security Group at Los Alamos National Laboratory in partnership with Oak Ridge National Laboratory. Wisdom and Sense was the second pass at an intrusion detection system for mainframes (the initial system, called ALAP, was fielded by the U.S. Department of Energy in several of the department's facilities). Wisdom and Sense operated on a UNIX platform and analyzed audit data from Digital Equipment Corporation VAX/VMS systems. Wisdom and Sense performed statistical, rule-based analyses that were quite different from other systems of the time. The system used *nonparametric techniques* (which are statistical techniques that make no assumptions about the distribution of the data) to derive its own rulebase from archival audit data. Wisdom and Sense then compared subsequent activity to this rulebase, looking for exceptions. The rulebase was structured into arrays of tree structures (called "forests"), and because the rules were "human readable," they could be "pruned" and modified by humans. These rulebases defined normal behavior as observed from the historical audit data.

The approach used by Wisdom and Sense was originally developed for a process control environment to monitor the transfer of nuclear materials within a Department of Energy test facility. Problems observed in Wisdom and Sense were similar to those observed in many machine-learning approaches of this era:

- Generating reliable learning sets of audit data that were known *not* to contain intrusions was extremely difficult.
- False alarm rates were high.
- The memory required to accommodate the huge rulebases was difficult to manage, making the prototype systems unstable.

Principal investigators for Wisdom and Sense were Hank Vaccaro, of Los Alamos National Lab; and Gunar Liepins, of Oak Ridge National Lab.

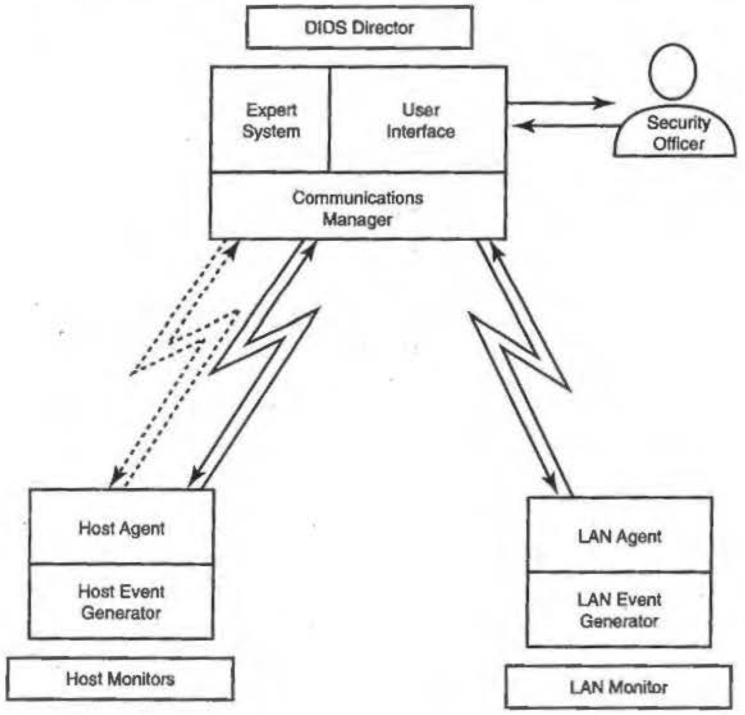
1.2.4 Integrating Host and Network-Based Intrusion Detection

Until 1990, intrusion detection systems were largely host-based, confining their examination of activity to operating system audit trails or other host-centric information sources. As noted in the preceding section, the NSM extended intrusion detection to the network environment. At the same time, drastic increases in the interconnectivity of systems (due to the growth of the Internet and the increase in computing and communications bandwidth) resulted in equally drastic increases in computer security concerns. The Internet worm of 1988 brought this concern to a fevered pitch,[17] and funding increased for both commercial and academic research and development efforts. The first major initiative to integrate host and network-based monitoring approaches was the Distributed Intrusion Detection System (DIDS).

The DIDS effort was a large-scale collaboration between the United States Air Force Cryptologic Support Center; Lawrence Livermore National Laboratory; University of California, Davis; and Haystack Laboratories.[18] The research was funded by the U.S. Air Force, the National Security Agency, and the Department of Energy. It was the first attempt to integrate host and network intrusion detection capabilities so that a centralized security management group could track security violations and intrusions across networks. The primary architect for DIDS was Steve Smaha.

The initial concept of DIDS was to use techniques (previously demonstrated in Haystack and NSM at host and network level, respectively) that centralized control and reporting in a DIDS central controller. Figure 1.5 illustrates the DIDS architecture.

Figure 1.5 DIDS Architecture



DIDS addressed several problems. One, a pressing issue in large network complexes, was tracking network users and filês across the network environment. This function was especially critical, given two factors. First, network intruders typically use the interconnectivity of different computer systems to hide their true identity and location. Some intruders, in fact, mount distributed attacks, in which each stage of an attack is sent from a different system. Second, perhaps the most durable cure for a network attack is to discover the person responsible for the attack, collect evidence of the person's responsibility for the attack, and then use law enforcement and the legal process to prosecute the person. DIDS was the first system to allow customers to deal with this problem in this context by automating the tracking and correlation of user identities across the monitored network.

For example, suppose someone on my corporate system decides to "network hop" through my system, en route to attacking my payroll server. Fortunately, I'm running DIDS on my network. By tracking the attacker's identity as he assumes user identities of Smith on host1, Jones on host2, and Bace on host3 (at which point the intruder attacks the file server for the payroll system), DIDS allows investigators to see that the person they need to talk to is actually the person sitting at the terminal associated with Smith on host1, not me!

Another problem addressed in DIDS was how one could correlate data from events happening at various layers of abstraction in the system. Such information is required to "see" problems as they affect the entire network. DIDS correlates the data by using a six-layer intrusion detection model, with each layer representing the results of a transformation applied to the data.

1.2.5 The Advent of Commercial Products

In the late 1980s to 1990, several organizations built intrusion detection tools—some in an attempt to capture early interest in this new security technology, others to support higher levels of trusted systems evaluations by the NCSC. Three of the earliest of these systems are discussed here. Others are covered in more depth in chapters that discuss specific strategies and lessons learned.

1.2.5.1 ComputerWatch

ComputerWatch is an audit trail analysis tool that was developed by AT&T in the late 1980s to provide audit trail analysis and limited intrusion detection capability. Data reduction was supported with an examination mechanism that provided different views of the audit data. These views were specified based on information relationships. For instance, one could see the sequence of events associated with a particular nser or group of users. Similarly, one could see all the events that occurred during a particular time interval. Finally, one could see all the events that involved a particular file or part of the system.

ComputerWatch supported a variety of queries and reporting features designed for a system security officer. An expert system was used to summarize system security-relevant events, and a statistical analyzer and query mechanism allowed statistical characterization of system-wide events. The system was designed to consume operating system audit trails generated by UNIX System V/MLS.[19]

After a brief period of availability as a commercial tool, AT&T withdrew ComputerWatch as a turnkey tool, using it as an internal resource in its consulting services group. Cherie Dowell was program lead for the ComputerWatch effort.

1.2.5.2 ISOA

The Information Security Officer's Assistant (ISOA) was developed by PRC, Inc. as a real-time security monitor. It was implemented on a UNIX workstation and supported automated as well as interactive audit trail reduction and analysis. ISOA used a set of thresholds and indicators to spot deviations from normal or expected behaviors, using a hierarchical scheme for correlating the indicators with levels of concern or suspicion. Detection was done both in-stream, in real time with discrete audit event checks; and at the end of sessions, with a threshold check of session statistics. The principal architect for ISOA was Vic Winkler.[20]

1.2.5.3 Clyde VAX Audit

Audit is a product that was developed by Clyde Digital (later RAXCO, and then Axent) to scan audit trails generated by Digital Equipment Corporation's VAX/VMS operating systems. Audit filters audit trails against 14 indications of possible security problems, including such things as dial-up sessions (indicating outsider access), sessions that occur after business hours, and sessions indicating file system browsing. Audit assigns weights and scores correlating to each of these indicators, and then provides summary reports that feature scores for each user on the system (where a high score indicates a high-risk user).[21]

Note

Allan Clyde, with sons Robert and Stephen, performed independent research and development in the mid-1980s on a surveillance-based security kernel for secure systems. These security kernels were built around surveillance gates (called *S-gates*), which were small software modules that monitored data paths within the operating system. The S-gates captured information from these data paths and processed it by using a weighted scoring analysis. The system then reported the results in a "suspicious event report."

The Clydes were notable for two reasons. First, they were building and fielding commercial products that were far ahead of most others in the security-monitoring field. Robert Clyde developed a product in 1977, called Control,[22] which allowed a system manager to monitor another user's terminal session for security reasons. He also wrote the first version of Audit, built on the S-Kernel model in 1983. Both of these products were developed for the Digital Equipment Corporation PDP-11 and VAX operating system environments. Second, the Clydes performed their research and development without government funding, building a commercially successful product organization that remains a major player in intrusion detection.

1.3 Conclusion

This chapter drew a timeline associated with audit, security audit, audit reduction, and intrusion detection. We outlined the concepts and strategies that are relevant to intrusion detection, and pointed out the seminal works for the area. It is important to note that the requirements and functions for these systems have changed significantly over the last 20 years as the nature of computing technologies and environments has changed.

The need for intrusion detection is even greater today than it was in the early 1980s when the seminal papers were written and initial research studies done. Developers of the next generation of intrusion detection products should be cognizant of the exploration done and the approaches tried in decades past. We'll revisit many of the systems cited in this chapter as we discuss the features and strategies for intrusion detection throughout this book.

Endnotes

1. Wasserman, Joseph J. "The Vanishing Trail." *Bell Telephone Magazine*, 47, no. 4, July/August 1968.
2. Abbott, Robert P. Personal communication, May 1999.
3. National Computer Security Center. *Glossary of Computer Security Terms*. Version 1, Rainbow Series, October 1988.
4. National Computer Security Center. *Department Of Defense Trusted Computer System Evaluation Criteria*. Orange Book, DOD 5200.28-std, December 1985.
5. National Computer Security Center. *A Guide to Understanding Audit in Trusted Systems*. Version 2, June 1988.
6. Anderson, James P. *Computer Security Technology Planning Study* 2. ESD-TR-73-51, Bedford, MA: Electronic Systems Division, Air Force Systems Command, Hanscom Field, October 1972.
7. Anderson, James P. *Computer Security Threat Monitoring and Surveillance*. Washington, PA: James P. Anderson Co., 1980.
8. Denning, Dorothy. "An Intrusion Detection Model." Proceedings of the Seventh IEEE Symposium on Security and Privacy, May 1986: 119–131.
9. Lunt, Teresa F. et al. "IDES: A Progress Report." Proceedings of the Sixth Annual Computer Security Applications Conference, Tucson, AZ, December 1990.
10. Halme, Lawrence, T. Lunt, and J. Van Horne. "Automated Analysis of Computer System Audit Trails for Security Purposes." Proceedings of the National Computer Security Conference, Washington, D.C., September 1986.

11. Tener, William T. "Discovery: An Expert System in the Commercial Data Security Environment." Proceedings of the IFIP Security Conference, Monte Carlo, 1986.

12. Smaha, Stephen E. "An Intrusion Detection System for the Air Force." Proceedings of the Fourth Aerospace Computer Security Applications Conference, Orlando, FL, December 1988.

13. Sebring, Michael M., E. Shellhouse, M. E. Hanna, and R. A. Whitehurst. "Expert Systems in Intrusion Detection: A Case Study." Proceedings of the Eleventh National Computer Security Conference, Washington, D.C., October 1988.

14. Hochberg, J. et al. "NADIR, An Automated System for Detecting Network Intrusion and Misuse." *Computers and Security* 12, no. 3, May 1993: 235–248.

15. Heberlein, L. T. "A Network Security Monitor." Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1990: 296–304.

16. Vaccaro, H. S. and G. E. Liepins. "Detection of Anomalous Computer Session Activity." Proceedings of the 1989 IEEE Symposium on Research in Security and Privacy, Oakland, CA, May 1989: 280-289.

17. Spafford, Eugene H. *The Internal Worm: Crisis and Aftermath.* Communications of the ACM, 32(6), June 1989: 678-687.

18. Snapp, S. R. et al. "DIDS (Distributed Intrusion Detection System)—Motivation, Architecture, and An Early Prototype." Proceedings of the Fifteenth National Computer Security Conference, Baltimore, MD, October 1992.

19. Dowell, Cheri and P. Ramstedt. "The ComputerWatch Data Reduction Tool." Proceedings of the Thirteenth National Computer Security Conference, Washington, DC, October 1990.

20. Winkler, J. R. "A UNIX Prototype for Intrusion and Anomaly Detection in Secure Networks." Proceedings of the Thirteenth National Computer Security Conference, Washington, DC, October 1990: 115-124.

21. Clyde, Allan R. *Insider Threat Identification Systems.* Rockville, MD: A. R. Clyde Associates, September 1987.

22. Clyde, Robert. Personal communication, June 1999.

CHAPTER 2

Concepts and Definitions

2.1 An Introduction to Intrusion Detection

Intrusion detection is the process of monitoring computer networks and systems for violations of security policy. In the simplest terms, intrusion detection systems consist of three functional components:

- An information source that provides a stream of event records
- An analysis engine that finds signs of intrusions
- A response component that generates reactions based on the outcome of the analysis engine

We'll flesh out this model as we proceed through this chapter.

Intrusion detection is an incarnation of the traditional practice of system audit. *Audit* is defined as "the official systematic examination of accounts to ascertain their accuracy."[1] Intrusion detection augments the traditional audit, which was designed to occur at infrequent intervals, thns making it a continuous process.

To better understand the core process of intrusion detection, let's look at the historical process from which it evolved. In computer security circles, *auditing* systems meant manually reviewing audit trails generated by computer operating systems and other system-logging mechanisms. This security audit trail review was structured to allow responsible parties to ensure that the activities that occurred on the computer system were in compliance with some set of security policies. Where the activities were found not to be in compliance, there were additional goals:

- Accountability—Determine who was responsible for the breach
- Damage assessment—Determine what actions they took and what damage ensued

- Damage recovery—Determine what steps are required to repair the damage and restore the system to secure operation

These traditional audit principles and practices are discussed in more detail in Chapter 1, "The History of Intrusion Detection."

As computer systems became faster, more complex, and more numerous, the size and complexity of audit trails increased as well. The task of reviewing the data grew much more onerous and then simply became impossible. Automating the process of audit review was a logical remedy for this problem. The requirement for an automated audit trail reduction and review function was included as part of the National Computer Security Center's Trusted Systems Security Criteria of the 1980s. Research done in the course of satisfying this automated audit requirement yielded much of what we know about modern security audit and intrusion detection technology.

2.2 Security Concepts

Intrusion detection was initially proposed and is constantly evolving to meet a set of functional goals, all associated with improving the security of computers and networks. In this section, we'll define some of the fundamental terms and concepts of computer and network security. This information will allow you to look at current and future intrusion detection strategies, and to ask yourself the following questions, which remain constant, even as the technology changes:

What security goals do I need to support?

What assumptions should I make about the security goals of the target system?

Exactly what assets do I need to protect?

Answering these questions (and more important, by noting which questions remain unanswered) will allow you to make intelligent judgments about the value of a proposed intrusion detection approach.

2.2.1 A Cultural View of Computer and Network Security

So what do we mean by *security*? Security can be viewed from two general vantage points: theoretical and engineering. Some practitioners of computer and network security approach the problem from an abjectly theoretical point of view. These security experts explore the theoretical foundations of computing and consider security in that context. They are interested in characterizing security properties mathematically by forming security models that are provably correct. The precision and clarity of view that these experts bring to the area can be quite valuable.

Another faction in the computer security world approaches security from a more pragmatic, engineering point of view. These security experts, although often interested in the etiology of security problems, are much more concerned with the questions of securing operational systems so that they can survive in the here and now. One might argue that all practitioners should be as technically rigorous as the theoreticians; one could also argue that all theoreticians should be able to administer operational systems!

Both of these approaches to computer security are legitimate, although purists in each group profess disdain for the other. Let's consider the fundamentals from both vantage points because each has something to offer to those interested in securing systems.

2.2.2 Practical Definition of Computer Security

A practical definition of a *secure computer system* is "a system that can be depended upon to behave as it is expected to."[2] From this intuitive view of security, we can infer the fundamental concepts associated with security. For instance, the notion of depending on a system implies that we *trust* that system. Is this trust quantifiable? If so, how do we measure it? Do we trust the system to behave as it is expected to? Who determines the expectations for system behavior? How do we determine whether the actual behavior of the system in fact matches the expected behavior?

2.2.3 Formal Definition of Computer Security

A more precise definition of security is given in terms of the "security triad": confidentiality, integrity, and availability.

C: *Confidentiality* is the requirement that access to information be restricted to only those users authorized for that access. Much of the work done by the government in computer security focuses on confidentiality.

An example of a system with goals of confidentiality is a banking system. As the customer of a bank, you expect the bank to protect your account information. Would you continue to do business with a bank if you discovered that your account records were accessible to other account holders or to the general public?

I: *Integrity* is the requirement that information be protected from alteration. Integrity is especially critical in systems handling data such as medical records (imagine the impact of someone altering doctors' orders on a patient record) or financial accounts. Many publicized Web site attacks involve breaches of integrity, in which address tables or site content are modified.

A: *Availability* is the requirement that the information and system resources continue to work, and that authorized users be able to access resources when they need them, where they need them, and in the form in which they need them. Many network-based attacks, such as "teardrop" and "ping of death," crash servers by sending them network traffic fashioned to exploit vulnerabilities in the operating system software running on those servers. These intrusions, which violate availability requirements, are labeled *denial of service* attacks.

A secure computer system supports all three goals of the security triad. In other words, a secure system protects its information and computing resources from unauthorized access, tampering, and denial of service.

2.2.4 Trust

If any central concept is associated with security, that concept is trust. *Trust* is the confidence that what is expected of a system entity corresponds to actual behavior. The level of trust corresponds to the level of confidence in this association between expected and actual behavior. System elements that interoperate do so with some assumption about the trust with which the other element is imbued. In cases where these trust assumptions prove unwarranted, vulnerabilities exist, and threats often follow.

In assessing trust relationships, you limit the area of concern by drawing a security boundary or perimeter. This approach allows a systematic assessment of trust relationships at each juncture, which yields considerable insight into the trustworthiness of the system.

Note that the question of trust arises, regardless of where you draw the security boundary of the system. You must be able to trust the administrators and users of the system not to abuse their privileges. You must also trust the environment in which the system physically resides to protect the system from physical hazards.

2.2.5 Threat

What determines the content of a security policy? Most security programs are driven by a desire to address a threat. A *threat* is defined as any situation or event that has the potential to harm a system. This harm can be in the form of disclosure, destruction, or modification of data; or denial of access to data or to the system-processing resources. Major categories of threat include hackers, viruses, fire, flood, lightning strikes—the list goes on and on.

From this list, you may have noticed that threats can be either internal or external to the system and can be intentional or incidental. To achieve security goals, you must consider physical threats as well as computer threats. You might also want to require background investigations of personnel serving in critical roles (such as system administrators) when they have significant control over computing and information resources.

How are threats structured in the computer security world? There are several ways to classify threat, and some involve the source of the threat. An early model specified the following three categories[3]:

- External penetrators—Unauthorized users of the system
- Internal penetrators—Authorized users of the system who overstep their legitimate access rights. These internal threats are divided into the following:
  - Masqueraders—Those who appropriate the identification and authorization credentials of others
  - Clandestine users—Those who successfully evade audit and monitoring measures
- Misfeasors—Authorized users who exceed their privileges

In intrusion detection, we build on this model of threat, using the term *intrusion* to mean any intentional violation of the security policy of a system. This definition encompasses all the threats covered in Anderson's model, plus other threats to system security not covered in his model. These threats include the following:

- People who attempt to gain access to a system or data
- Programmatic threats (software attacks such as viruses, trojan horses, and malicious Java or ActiveX applets)
- People who probe or scan systems in search of vulnerabilities they can exploit in a later attack

2.2.6 Vulnerability

Security problems in computer systems result from vulnerabilities. *Vulnerabilities* are weaknesses in systems that can be exploited in ways that violate security policy. Vulnerabilities occur in a multitude of ways. For example, weaknesses occur in the design and implementation of the system software and hardware. These weaknesses are sometimes called *technical vulnerabilities*. Other weaknesses occur in security policy, procedures, controls, configuration, or other system management areas. These fall in the realm of *procedural or management vulnerabilities*.

Several rules of thumb govern the likelihood of vulnerabilities occurring in systems. The larger the system, the greater the likelihood of vulnerabilities. The more complex the system, the greater the likelihood of vulnerabilities. The more dynamic the system and its environment (for example, the more often a system is updated or replaced with a new system), the greater the likelihood of vulnerabilities.

Although threat and vulnerabilities are intrinsically related, they are not the same. Threat is the result of exploiting one or more vulnerabilities. Intrusion detection is designed to identify and respond to both.

2.2.7 *Security Policy*

Security policy is required in order to map the sometimes-arcane concepts of security to the real world. In our initial definition of security, we pointed out that it is based on some notion of what constitutes expected behavior for a system. Security policy documents these expectations.

2.2.7.1 *Procedural*

Security policy has two common definitions. The term most often refers to the set of management statements that document an organization's philosophy of protecting its computing and information assets. This *procedural* or *managerial security policy* outlines security goals and commits management resources to meeting these goals. It also assigns responsibilities, defines roles, and establishes management and security controls. Finally, the policy sets up procedures and practices for securing information and computing assets. Figure 2.1 outlines the structure for a procedural or managerial security policy. Note that while policy remains consistent, the procedures are more specific, but they rarely change, and the practices are quite dynamic, reflecting the particulars of the system at the current time.

Figure 2.1 An Example of Procedural System Policy, Procedures, and Practices

Policy	Procedure or Standard	Practice or Guideline
"We will protect corporate systems from unauthorized alteration of software."	"Where applicable, virus checkers shall be run on corporate systems."	"Norton Anti-Virus will be run on all Windows 95/98 systems at least once a week. The software will be updated by MIS staff at least once a month."
	"Integrity-checking tools shall be used on critical system files and executables."	"Tripwire® will be run at least once per quarter and upon any update or alteration on all UNIX server systems directories, Checksums will be saved on removable media stored in accordance with corporate policy."

The goal of security policy for a computer system is analogous to the goal of legal codes in a society. Both seek to protect legitimate users of system resources from ill effects due to the activities of miscreants. Procedural security policies are usually informal; that is, they are written in ordinary language, not as mathematical expressions.

2.2.7.2 Formal

A *formal security policy* usually consists of a mathematical model of the system as a collection of all its possible states and operations, accompanied by a set of constraints on when and how the states and operations may exist. The government's Trusted Systems Initiative defines the security policy of the system as the set of rules enforced by the system's security features.[4] Writing security policies that formally and precisely define which activities are not allowed is a difficult job. Note that according to this definition, *every* system enforces an implicit security policy!

Formal security policies have certain advantages for those interested in performing intrusion detection. Such policies are structured and precise in a way that makes it easier to translate their intent into detection patterns. These policies can also provide guidance with regard to what information the system audit mechanism should collect for analysis.

2.2.8 Other Elements of the System Security Infrastructure

It is important to understand that intrusion detection is not, nor was it ever meant to be, a silver bullet for computer security (an infallible and complete system security solution). Many are tempted to consider it as such. It is, however, an integral part of a system security suite that in totality protects a computer system from intrusion and internal abuse.

Consider intrusion detection in the context of physical security. Suppose you have a valuable asset, you use physical security techniques and measures to protect it. For instance, you'd put the asset inside a building, under a well-sealed roof, and shielded from weather-related damage. You'd make sure that the building walls were constructed of strong materials—say, concrete block, not cardboard. The building would be designed with few windows and doors. Those openings would be located to minimize the possibility of a burglar using them as entry points. The asset might be placed inside a safe or vault, with a strong lock on the door. You might place fences, moats, or other barriers around the building. If you were particularly paranoid about protecting the asset, you might hire armed security guards and charge them with monitoring the premises, logging any entry to the building, and patrolling the building and grounds on a given schedule. Finally, you might install sophisticated burglar alarms and surveillance systems to provide additional assurance that the asset is protected. You see this sort of protection scheme implemented every day in banks, military bases, and art museums.

Although the burglar alarm and surveillance systems undoubtedly offer a great deal of protection to the asset, can they replace everything else in the protection suite? Of course not! The combination of protective measures function in concert to protect your asset. Together, they yield much more robust and cost-effective protection than pouring all your resources into any one of them alone.

This applies to the computer and network world, as well. Many components and functions serve as part of a sound system protection strategy. Some of these protection mechanisms are discussed in the following sections.

2.2.8.1 Access Control

Access control mechanisms are responsible for restricting access to objects according to the access rights of the subject. Access control is divided into Mandatory Access Control (MAC), in which decisions about access rights are embedded in the system; and Discretionary Access Control (DAC), in which the owner of an object sets up and controls access rights for that object. In systems offering MAC, the access rights are determined by security labels on the objects. Some commercial products offer encryption-based solutions that allow customers to formulate and enforce DAC policies.

2.2.8.2 Identification and Authentication

Identification and authentication mechanisms (I&A) support the positive identification of subjects and objects to the system.

Authentication mechanisms can be divided into three categories, depending on what they require of you: what you know, what you have, and what you are. Each category involves the system challenging a user for a secret that is known only to the user and the system. If the secret presented by the user matches the secret held by the system, the system validates the user's identity and allows access to the system.

"What you know" is the basis for the traditional I&A mechanism, found in most operating systems, in which a user login and password challenge identifies and authenticates users. Unfortunately, this approach has proven to be vulnerable to a variety of attacks, such as password-cracking and trojan horse password-grabbers. These attacks result in the exposure of the secret (the password) to adversaries. This authentication technique is gradually being replaced by stronger, zero-knowledge techniques that avoid passing the secret itself.

Examples of authentication systems based on "what you have" include token-based systems, such as those in which the user is provided with a smart card, a key, or a special disk. Many of these tokens are designed to use cryptographic mechanisms and physical tamper-resistance mechanisms to prevent attackers from counterfeiting or spoofing them.

Examples of systems based on "what you are" include biometric schemes, which use the voice, fingerprints, or retina of the user to identify and authenticate the user to the system. Some systems use hybrid approaches; for example, one commercial smart-card token includes a fingerprint scanner.

The underlying challenge-response authentication process is sometimes used for security purposes other than commencing system access. The process can also be used in intrusion detection as a means of establishing whether suspicious behavior is originating from an intruder or a legitimate user. This topic is discussed further in Chapter 5, "Responses," in which we discuss system responses to detected problems.

2.2.8.3 Encryption

Perhaps the oldest information security mechanism, encryption, safeguards information by performing a variety of functions. It can effectively obscure the content of data files or transmissions, eliminating the possibility of surveillance by unauthorized parties. It can also detect accidental or intentional alterations in data. Encryption can provide verification that the author or originator of a document is the person you expect.

Encryption is the process of taking an unencrypted message (called *plaintext*), applying a mathematical function to it (*encryption algorithm*) with a key, and producing an encrypted message (called *ciphertext*). Although encryption provides powerful protection of information, it is not perfect. It cannot prevent intentional deletion of encrypted data, for example. It cannot protect data before it is encrypted nor after it is decrypted. And the encryption is dependent on the key being protected. Should the key be guessed or divulged, the value of the encryption is nullified.

2.2.8.4 Firewalls

Firewalls provide a security boundary between networks of differing trust or security levels by enforcing a network-level access control policy. The mechanisms used include proxy servers, network packet filters, and encrypted data tunnels (also called Virtual Private Networks). Firewalls filter network packets, making decisions to allow and disallow passage of packets according to a specified policy. Firewalls also allow an address translation for networks so that the configuration details of an internal network can be hidden from potential intruders.

2.2.9 How Security Problems Occur

When security problems occur, especially given the ramifications of the problems, you might wonder why they exist. Although the number of specific problems is huge, the etiologies of the vast majority of security problems fall into three categories: design/development, management, and trust.

2.2.9.1 *Problems in Design/Development*

The first problem consists of errors, flaws, and omissions that occur in the design and development of systems. These result in vulnerabilities in both software and hardware. For instance, researchers have discovered that cryptographic keys, burned into smart-card hardware to protect them from disclosure, can be extracted by a process in which faults are injected into the cards by varying the operating voltages or clock cycles. Another classic example is the problem of race conditions in system software. Race conditions result when an interval occurs between the time a value is checked for validity and the time the value is actually used; in the interval, an adversary can substitute an illicit value and dupe the software routine into performing a nonstandard operation. Yet another vulnerability, which accounts for many known UNIX attacks, is the failure to check the arguments that are passed to privileged programs from the command line. Attackers create problems by invoking these programs, passing them arguments that overflow the input buffers, and thereby crashing the program and allowing the attacker a privileged shell.

Many problems in this category can be prevented by rigorous application of sound engineering practice, including quality assurance.

2.2.9.2 *Problems in Management*

The second problem area that leads to security problems is that of managing fielded systems. This problem encompasses errors made when configuring the system itself or security systems that are meant to provide protections to the system. An example is using inappropriate privilege settings for system files (such as making password files for UNIX systems readable and writeable by "world"). This problem area also includes scenarios in which system administrators or users disable or circumvent security mechanisms. A problem that commonly occurs in large organizations is that of users who install unauthorized modems on internal systems—desktop PCs, for example. Because the modems provide a path into the organizational internal networks, they allow adversaries to circumvent the protection afforded by the firewall!

2.2.9.3 *Problems in Trust*

Perhaps the most pervasive problem is that of naivete concerning appropriate assumptions regarding trust. Many of these occur because of an unforeseen differential between the development environment and the operational environment. For example, UNIX was originally developed by programmers in a collegial environment. The information sharing was standard and threat levels were low. With the passage of time, however, UNIX was fielded in commercial settings, where the threat model is different. In this case, trust assumptions did not generalize to environments beyond the original development site.

In fact, some problems in design and development actually apply here, too. Designers and implementers of systems trust that the system will be used in a certain manner within a certain context. The designer trusts the customer to use a system in a particular fashion. The customer trusts the product to perform as promised. In extensive systems, the users trust all parts of the system to function as expected. This trust extends to the human portions of the system as well. Users trust that someone is administering the system in a competent, consistent fashion.

So, what happens when this trust is breached? This situation usually means that a security problem has occurred. In many cases, the problem is transparent to everyone but the person responsible for the problem unless, of course, someone is monitoring the operation of the system and notices this unusual system activity. This topic is covered in greater depth in Chapter 8, "Understanding the Real-World Challenge."

2.3 Intrusion Detection Concepts

As intrusion detection evolved over the past 20 years, so did strategies for tackling the issues associated with intelligently monitoring systems for problems. This section outlines these strategies and the respective rationale that underlies each one. In so doing, I hope to provide you with an understanding of key concepts. This information will allow you to assess the strengths and weaknesses of future intrusion detection approaches as they are proposed..

2.3.1 Architecture

When audit was proposed as a protection for sensitive systems, it was apparent that for audit information to be trusted, it must be stored and processed in an environment separate from the system that it protected. This requirement was inherited by most intrusion detection approaches. Separating audit information from the system that the audit is protecting is necessary for three reasons:

- To keep a successful intruder from disabling the intrusion detection system by deleting audit records
- To keep a successful intruder from modifying the results of the intrusion detector to hide the presence of the intrusion
- To lessen the performance load associated with running intrusion detection tasks on an operational system

In this architecture, the system running the intrusion detection system is called the *host*, and the system or network being monitored is called the *target*.

2.3.2 Monitoring Strategy

The first requirement for intrusion detection is a data source. This element can also be considered an event generator. Data sources can be categorized in a variety of ways. For intrusion detection purposes, we first categorize them by location. This classification scheme divides system-monitoring views into four categories: host, network, application, and target. We will use the term *monitoring* to describe the action of collecting data from a data source and passing it to an analysis engine.

- *Host-based monitors* collect data from sources internal to a computer, usually at the operating system level. These sources can include operating system audit trails and system logs.
- *Network-based monitors* collect network packets. This is usually done by using network devices that are set to promiscuous mode. (A network device operating in promiscuous mode captures all network traffic accessible to it, not just that addressed to it.)
- *Application-based monitors* collect data from running applications. The data sources include application event logs and other data stores internal to the application.
- *Target-based monitors* function a bit differently from the other monitors listed in this section because target-based monitors generate their own data. Target-based monitors use cryptographic hash functions to detect alterations to system objects and then compare these alterations to a policy. Because the state of the target object is monitored versus the activity taking place on the system housing the object, this monitoring strategy can be efficient for some systems that cannot be monitored using other approaches.

2.3.3 Analysis Type

In the intrusion detection process, after information sources are defined and locations are established, the next requirement is an analysis engine. This system component takes information from the data source and examines the data for symptoms of attack or other policy violations.

In intrusion detection, most analysis approaches involve misuse detection, anomaly detection, or some mix of the two.

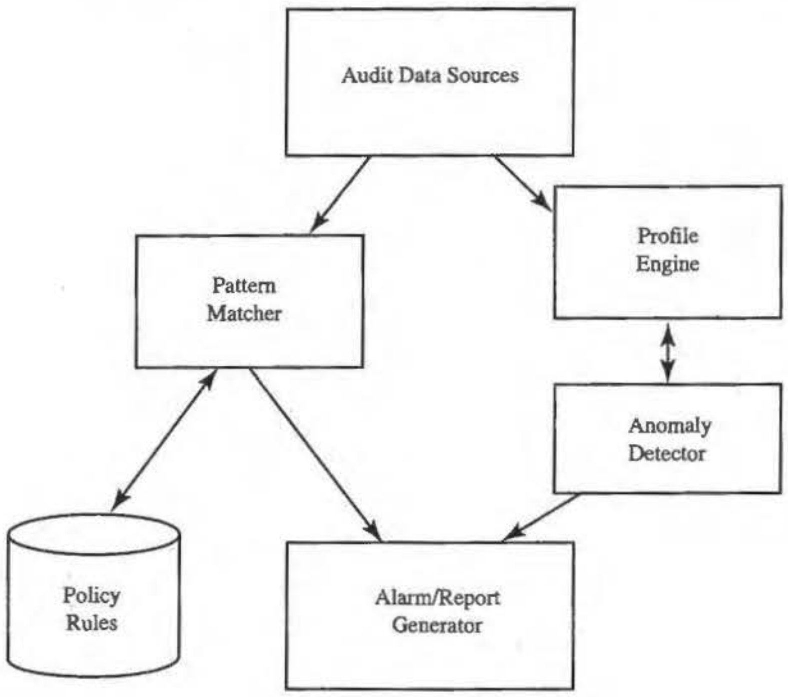
- *Misuse detection*—Engines look for something defined to be "bad." To do this, they filter event streams, searching for activity patterns that match a known attack or other violation of security policy. Misuse detection uses pattern-matching techniques, matching against patterns of activity known to indicate problems. Most current, commercial intrusion detection systems utilize misuse detection techniques.

- *Anomaly detection*—Engines look for something rare or unusual. They analyze system event streams, using statistical techniques to find patterns of activity that appear to be abnormal. This approach reflects the view of some intrusion detection researchers that intrusions are some subset of anomalous activity.

Significant advantages are associated with combining the two analysis schemes. The anomaly detection engine allows the system to detect new or unknown attacks, or other scenarios of concern. The misuse detection engine protects the integrity of the anomaly detection engine by ensuring that a patient adversary cannot gradually change behavior patterns to retrain the anomaly detector to accept attack behavior as normal.

Figure 2.2 shows a diagram of a typical intrusion detection system that uses both anomaly detection and misuse detection analysis approaches.

Figure 2.2 A Generic Intrusion Detection System



2.3.4 Timing

Another differential in analysis is the timing of the analysis of the data. Analysis can be done in a batch mode (known by some as "interval based" mode) or in real time.

2.3.4.1 Interval/Batch

Batch mode analysis means that the information source is conveyed to the analyzer in a file and that the information for a particular period of time is processed, with the results returned to the user after an intrusion takes place. Batch mode was the prevalent model for early intrusion detection because the communications and processing bandwidth of early systems did not support real-time monitoring or detection.

2.3.4.2 Real Time

As system speed and communications bandwidth grew, more intrusion detection systems moved to real-time analysis. In real-time analysis, the information source is conveyed to the analysis engine as the event happens (or with miniscule delay), and the information is processed immediately. We use the term *real-time* in the process control context, referring to a process that is fast enough to allow the results of the intrusion detection system to affect the progress or outcome of any intrusion it sees.

Note

For many years, the intrusion detection community engaged in a lively debate, in which the term *near real time* was coined to refer to intrusion detection that involved processing a continuous stream of events but not at blinding speed. In many analysis approaches, modern machine speeds have rendered that debate moot.

Real-time analysis approaches have enabled automated responses to intrusions. This development has considerable appeal to many customers.

2.3.5 Goals of Detection

Another factor in intrusion detection analysis is the goal the detector supports. The two traditional goals driving intrusion detection are accountability and active response.

The goals for intrusion detection affect a number of design and implementation decisions. For instance, if the goal is to establish user accountability for detected intrusions, one might need to maintain raw audit data to support legal requirements for prosecution. Suppose, on the other hand, one doesn't care about the identity of the intruder but is more interested in blocking the attack and then correcting the vulnerability that allowed it. In this case, after the data is analyzed and the intrusion is detected, the audit data can be discarded.

2.3.5.1 *Accountability*

Accountability is the capability to map a given activity or event back to the responsible party. Because the ultimate goal of establishing accountability is to extract some compensation or pursue some legal remedy against that responsible party, it helps if that responsible party is a human (not a machine name). Furthermore it is even more helpful if a physical address or other link to the physical world can be associated with that party.

Maintaining accountability on networks presents one of the major challenges in intrusion detection. Given current network schemes, when an attack utilizes programs and daemons running on separate machines, separate sets of identities are associated with each host involved. In general, the greater the number of hosts involved in a network attack path, the lower the probability of successfully detecting an attack and establishing accountability.

One of the earliest attempts to integrate host-based and network-based intrusion detection capabilities, the Distributed Intrusion Detection System (DIDS) was designed to enable a major customer to maintain user accountability on its large internal network complex. The customer also wanted DIDS to allow the tracking of related events across the network as a whole. The DIDS team approached this problem by creating a user network identifier (called a NID) the first time a user entered the environment monitored by DIDS and then mapping all subsequent activities of that user to that NID.[4]

2.3.5.2 *Response*

In intrusion detection, *response* occurs when analysis yields an actionable result. Response is not limited to taking action against a suspected attacker, although this option captures the imagination of many in the network security world.

Perhaps the most common response is to record the results of the analysis in a log file, using that file to generate a report. This information is helpful to a variety of people in the corporate management structure. Some report features allow automatic report generation, with options for providing findings at different levels of detail for different reporting audiences. For example, the report that goes to the CIO might list the numbers of intrusions detected and the severity of those intrusions. The report that goes to the system manager would have information about the intrusion, the files that the intruder touched, the vulnerability probably exploited, and whether any fixes are available for it.

A more immediate response option is to trigger alarms of a variety of predetermined types. These might include an alarm flag on a network manager's console, a message to a security manager's pager, or, in certain instances, an email message sent to a system administrator.

Another response is to modify the intrusion detection system or the target system. Modifying the intrusion detection system can involve changing the information collected by the monitors or changing the character of the analysis performed on the information stream. Modifying the target system can include changing configuration settings for critical files to block the progress of a known attack. This ability to adapt the system in response to a suspicion level is quite valuable.

The remaining response option receives a great deal of attention from the press, the military, and the science fiction writing community. This is the strike-back response in which one blocks the intrusion, sometimes mirroring the attack back to its ostensible source. More benign active responses are also available, such as sending messages to firewalls or routers, which direct them to block subsequent network accesses from the source of the attack.

2.3.6 Control Issues

Another aspect of intrusion detection is controlling the system. There are two major approaches to controlling intrusion detection systems when the system monitors multiple hosts or networks.

2.3.6.1 Centralization

The first approach is a centralized reporting and control architecture. In this approach, a central node controls all the intrusion detection elements of a system. Centralization carries some requirements with it. For example, there must be a way to secure messages between system components. There must also be a way to determine whether an element of the system has been tampered with or disabled at a given time. In addition, centralization requires a way to gracefully start and stop system components. Finally, the centralized approach requires a way of consolidating status information and presenting it to end users in a form that is meaningful to them.

2.3.6.2 Integration with Network Management Tools

One way of addressing some of the issues associated with the centralized control of intrusion detection is to simply regard intrusion detection as a function of network management. Some system information streams collected by network management packages can be used as information sources for intrusion detection. In addition, some alarms raised by intrusion detection systems affect network availability and performance. Therefore, integrating the two functions can be of value to customer organizations. Many commercial intrusion detection packages offer the option of spawning Simple Network Management Protocol (SNMP) messages for network management tool capture.

2.3.7 *Determining Strategies for Intrusion Detection*

Given the multiple options associated with each process component, many possible strategies are available for tackling the problem of detecting attacks and other activities of interest. The optimal strategy often depends on factors such as the following:

- The criticality or sensitivity of the system being protected
- The nature of the system (for example, the complexity of the system hardware and software platforms)
- The nature of the organizational security policy
- The threat level of the environment in which the system operates

2.4 *Conclusion*

Intrusion detection systems have, from their outset, generated a great deal of attention in the security world. To many, they present a vision of an ever-vigilant system sentinel, equipped with the capability to assimilate great quantities of information generated by complex systems, infallibly finding security problems, tracing them back to the responsible parties, and then taking action to isolate and heal any damage that occurred.

The current promise of intrusion detection is all this, and more. The growth of the Internet and the subsequent push to place more and more critical information on network-connected systems creates a scenario in which security is of critical importance.

Traditional approaches to computer security (such as the National Computer Security Center's Trusted Systems Evaluation program) espouse an approach to security based strictly on the mathematically rigorous secure design of systems. The alternative to the secure design approach is an approach called (often derisively) "penetrate and patch."

Intrusion detection systems allow users to optimize the penetrate-and-patch process by sharing penetration knowledge between sites. Many users readily acknowledge that penetrate and patch is not an optimal solution to system security problems. However, it is often the only available option to users who must depend on the operating systems and software developed by commercial entities that have neither the capability nor the inclination to perform secure design. Given this state of affairs, intrusion detection systems, by augmenting the commercial exchange of attack information, may represent the only way of avoiding a descent into security chaos.

In addition, the event monitoring and attack recognition capabilities of intrusion detection systems enhance the security of a system in other ways. First, these capabilities have a significant deterrent effect on attackers, who run a greater risk of discovery and subsequent prosecution for their attacks. Furthermore, intrusion detection systems can deal with insider threat, a significant problem in system security. Second, automated responses can disrupt some attacks at the outset, making subsequent attacks more difficult to stage. Third, monitoring the operation of the rest of the security infrastructure allows system security managers to see when security protections are not functioning properly and to rectify the situation before an attacker exploits it. And fourth, the information gained by monitoring the system can sometimes make it easier to manage the system in other ways, resulting in greater system reliability.

Endnotes

1. *Oxford English Dictionary, Second Edition.*
2. Garfinkel, S. and E. H. Spafford. *Practical UNIX and Internet Security, Second Edition.* O'Reilly and Associates, 1996.
3. Anderson, "Computer Security Threat Monitoring and Surveillance." J.P. Anderson Co., 1980.
4. Ko, C., D. Frincke, T. Goan, L. T. Heberlein, K. Levitt, B. Mukherjee, and C. Wee. "Analysis of an Algoritham for Distributed Recognition and Accountability." Proceedings of the First ACM Conference on Computer and Communication Security, Fairfax, VA, November 1993: 154–164.

CHAPTER 3

Information Sources

"A computer, to print out a fact,
Will divide, multiply, and subtract.
But this output can be
No more than debris,
If the input was short of exact."
- Gigo

The first requirement for intrusion detection or any such data-driven processing task, is a set of input data. For intrusion detection, that input data comes from a variety of sources. In this chapter, I cover the data sources commonly used in intrusion detection and describe how they influence the capabilities of the intrusion detection system. I also discuss the issues associated with each of these data sources and identify possible strategies for dealing with them.

3.1 The Organization of this Chapter

This chapter covers three data categories: data derived from sources internal to individual systems (*host-based sources*), data derived from sources associated with the network (*network-based sources*), and data derived from other sources (out-of-band sources, such as telephone switches or physical security systems). I cover these sources in this order for several reasons. First, this sequence mirrors the path that computer technology in general, and intrusion detection specifically, has traveled over time: host-based centralized architectures, then network-based distributed models, and finally ubiquitous point products—each optimized to perform a particular function. Second, this approach moves from lower- to higher-level abstractions of the systems from which the data is collected.

Note

In this chapter, the generic term *monitor* refers to any information collection mechanism used by an intrusion detection system. The term *protection domain* refers to the area of the system that the intrusion detection system is meant to monitor and protect.

3.1.1 Which Source Is the Right Source?

So, your next question might be the following: Which of these sources is the best source for intrusion detection? The correct answer is that it depends on what you're interested in detecting. To detect an attack, the intrusion detection system must be able to "see" the evidence of that attack. Perhaps a subtler point is that to detect and then deal with the attack, the intrusion detection system must also be able to see the *right* data about the attack (for instance, the outcome of the attack).

For instance, suppose that an intrusion detection system detects an attack carried out by sending malformed network packets to a particular host. (This type of intrusion is a common form of network-based denial-of-service attack; a vulnerable target system responds to the attack by crashing.) A network-level monitor might capture the malformed packets and the fact that they were sent from a particular IP address to a particular host inside the protection domain. However, the addition of a host-level monitor can allow the intrusion detection system to see whether that malformed packet caused the system to crash. Furthermore, the addition of a network monitor on every network segment in the protection domain allows the intrusion detection system to see that 840 identical malformed packet attacks were launched against other hosts in the domain over the last six hours, and 300 of the attacks appear to have been launched from a single dial-up connection.

3.1.2 Enduring Questions

Several questions have persisted over the course of intrusion detection history. These include the following:

- How much information is enough to allow you to accurately diagnose security problems without crippling the systems you're trying to protect?
- How do you select the right information to collect, and from where should you collect it?
- How do you manage the information collected to support any legal remedies you might want to pursue against attackers?
- How do you honor your responsibility to handle the information collected about users so that you stay within legal, regulatory, and ethical policy limits?

- How can you format this data so that you can organize and make sense of a wide variety of system platforms? Being able to understand various platforms becomes the key to performing intrusion detection over complex systems of interest.

Keep these questions in mind as we explore the nature of information sources. There is more discussion in Chapter 7, "Technical Issues," in which we explore current technical issues; and in Chapter 9, "Legal Issues," in which we discuss the legal issues associated with intrusion detection.

3.2 Host-Based Information Sources

The intrusion detection product community is divided into several factions. Some define intrusion detection systems as strictly network-based systems. Others define intrusion detection systems as strictly host-based or as specific applications-based systems. Still others advocate integrated approaches to intrusion detection in which both host-based and network-based approaches are combined to improve detection performance.

These approaches are based on the *monitoring approach* of the system, that is, the point at which information is collected. This section considers host-based information sources and looks at the mechanisms that produce the information.

Host-based information sources consist of *operating system audit trails*—that is, records of system events generated by a specialized operating system mechanism—and *system logs*—files of system and application events, which are usually text files written a line at a time by system programs.

3.2.1 Operating System Audit Trails

The first host-based information source considered to be of security significance was the operating system audit trail. Operating system audit trails are generated by a specialized auditing subsystem included as part of the operating system software. These audit trails are a collection of information about system activities, which are placed in chronological order and then organized into one or more *audit files*. Each audit file is composed of *audit records*, each of which describes a single system event. These records are generated by user actions and by processes invoked on behalf of users, whenever either makes system calls or executes commands. The commands can be local or remote. Each audit record is composed of a series of *audit tokens*, each of which describes the fields within the record.

Many operating system audit trails were originally designed and developed to meet the requirements of the Trusted Product Evaluation Program. This U. S. government initiative produced evaluation criteria—most notably the Trusted Computer System Evaluation Criteria (TCSEC)—also known as the "Orange Book"—that outlined the features and assurances required of commercial operating systems and applications software that was to contain and process classified information. The evaluation process was structured to rate the operating systems according to "trust levels" that corresponded to the perceived trustworthiness of the system.

The audit requirements of the Orange Book represented a conundrum to the vendor and user community because there were extensive requirements for audit capabilities but no directions for actual audit utilization. The criteria outlined extensive lists of events that audit systems should be capable of monitoring, but offered no guidance for selecting those events. The requirements also neglected to define how to structure, store, and use the audit information that was generated.[1]

Consequently, vendors provided a crazy quilt of audit capabilities in an attempt to meet the letter of the C2 audit requirements. In extreme cases, intrusion detection developers who were adept at dealing with the audit trails generated by one operating system vendor would not be able to comprehend the audit trails produced by another. Furthermore, some implementations produced audit mechanisms that were not useful in operational environments, thanks to storage or performance costs. Indeed, some early intrusion detection research efforts in which I was involved found massive flaws in several C2-evaluated audit mechanisms that suggested that the mechanisms had not been tested before they were fielded.

3.2.2 Approaches to Structuring Audit Trails

Operating system vendors use at least two design approaches in their audit systems. One approach creates *self-contained* audit records, which do not require other records for interpretation. The other chooses to eliminate redundant information in audit trails by distributing the information about an event over multiple records. The former is easier to process and comprehend, but the latter is helpful when conserving storage space allocated to audit trails.

3.2.3 Problems with Commercial Audit Systems

Despite the fact that the intrusion detection community appears to be the major consumer of operating system audit trails, a recent study found that no major operating system vendor actually provides audit trails that support the needs of intrusion detection systems! Vendor documentation for audit subsystems is often missing. When the documentation exists, it rarely corresponds to the audit system, as built. Many audit trails generate irrelevant or insufficient information. These deficiencies have resulted in people suggesting significant amendments to audit systems. For instance, some have proposed that effective host-based intrusion detection may require the developer to amend operating system kernel code to generate event information. This approach extracts a cost in system performance, which might be unacceptable for customers running computationally greedy applications. Furthermore, appreciable costs are associated with the maintenance of these operating system software alterations.[2]

3.2.4 *Pros and Cons of Operating System Audit Trails*

Despite the problems outlined here, many intrusion detection experts consider operating system audit trails preferable to other common host-level information sources for intrusion detection purposes. The primary reason is that the operating system is often structured to provide substantive protection for the audit subsystem and the audit trails that the subsystem generates. This protection is of obvious interest, given the security goals of intrusion detection.

Another motivation for prizing operating system audit trails as a data source for intrusion detection is that they reveal system events at a finer-grained level of detail. This information allows the intrusion detection system to spot subtle patterns of misuse that would not be visible from a higher level of abstraction. This finer-grained level of detail also makes it more difficult for an adversary to successfully corrupt the audit process by inserting false audit records.

As one might expect, this finer level of detail comes at a price. Both the volume as well as the complexity of the audit data rise with greater detail. Ironically, as we already noted, although the level of detail makes it more difficult for an adversary to circumvent the audit process entirely, the greater volume and complexity of the data make it easier in practice for intruders to hide their footprints.

Collecting data at too coarse a level of detail presents problems, as well. In particular, this lack of detail makes it impossible to differentiate between system activity invoked directly by a user and activity invoked by a program that has taken on the user's identity.

This deficiency is significant. The concept that a system program can act as a user's agent without that user's permission represents a case for the "*on behalf of*" semantics implemented in UNIX and Windows NT operating systems. An entire class of system attack exploits the capability of programs to assume the identity of a user or other subject possessing greater privilege than the actual user. This behavior applies particularly to attacks that target SUID root processes in UNIX systems. To detect this class of attack, the data source for an intrusion detection system must provide data at a fine enough level of granularity to allow the intrusion detector to differentiate between user and process.

Another advantage of monitoring activity at a greater level of detail is that it highlights abnormal patterns of process execution. This monitoring detail enables the intrusion detection system to recognize the execution of trojan horses and other malicious code.

3.2.5 *Content of Audit Trails*

Most commercial operating systems record events at kernel level (reflecting system calls) and at user level (reflecting application events). Audit records contain information about subjects responsible for the event and any objects involved in the event. Most records also include information about the process that initiated the event, the userID associated with the event, which sometimes includes the current userID as well as the original userID (in case the user identity changes). Kernel-level entries contain system call arguments and return values, whereas user-level entries contain high-level descriptions of the event or application-specific data.

This section covers the audit trail structure and content of two major operating systems: Basic Security Module (BSM) from Sun and Windows NT from Microsoft.

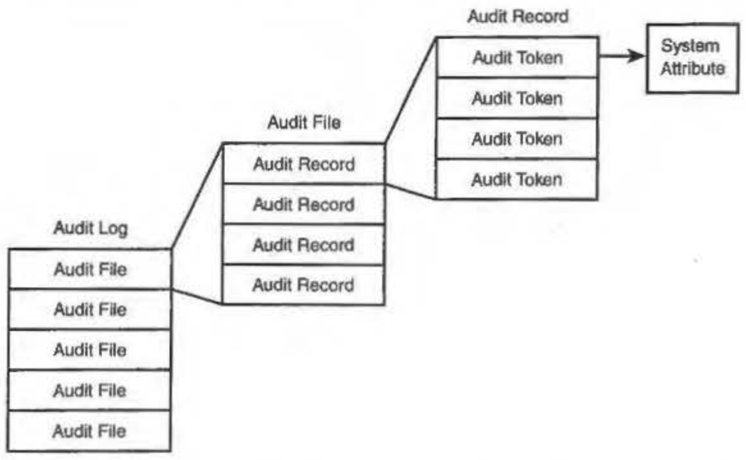
3.2.5.1 Operating System Example 1: Sun Solaris BSM

Sun's BSM security package is provided to bring Sun's UNIX operating systems into compliance with the TCSEC C2 trusted system rating. Although we focus on the auditing features of the package, it also provides device allocation mechanisms that meet C2 requirements for object reuse.

BSM Auditing Subsystem Structure

The BSM auditing subsystem, pictured in Figure 3.1, consists of an audit log, audit files, audit records, and audit tokens.

Figure 3.1 Structure of Sun BSM Audit Data



A BSM *audit log* consists of a sequence of *audit files*, which are in turn composed of *audit records*. Each audit record consists of a sequence of *audit tokens*, each of which describes a system attribute. Structures defining audit files have special file tokens to mark the beginning and end of files; header and trailer tokens delineate each audit record.

Audit records are described as either kernel-level or user-level generated records, depending on the nature of the event described in the record. As you might guess, kernel-level audit records are generated by kernel-level system calls; user-level records are generated by all other system calls.

BSM includes translation functions that help trace accountability for particular audit events. An event-to-system call translation translates each audit event to the kernel or user event that generated it, and an event-to-command translation translates each audit event to the application or command that generated it. These translations can be helpful in interpreting the contents of audit data.

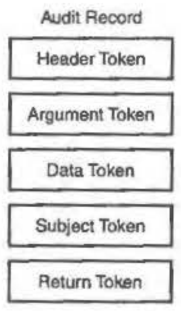
BSM audit records are generated and managed in binary form. Predetermined byte orders and data sizes facilitate cross-platform compatibility.

Events are grouped into *audit event classes* for audit management purposes. *Preselection* (selecting the system events for which audit records are generated) and *postselection* (selecting the audit records to extract from an audit log file) of audit events is done by specifying the applicable audit event classes.

BSM Audit Record Structure

Figure 3.2 shows the structure of a typical BSM audit record. Each auditable event in the system produces a particular type of audit record. Each audit record begins with a header token, which marks the beginning of the audit record in the audit trail. Most audit records contain a subject token, which refers to the process that caused the event. Finally, depending on audit policy, the record may end with a trailer token. For user-level and kernel events, the tokens describe the process that performed the event, the objects on which it was performed, and the objects' tokens, such as the owner or mode.

Figure 3.2 Structure of Typical Sun BSM Audit Record



Each user-level and kernel-level event has header, subject, and return tokens. In addition, many events also include a trailer token, but it is optional. Other optional tokens include the group and sequence tokens. Use of these tokens depends on current audit policy.

Each token begins with a 1-byte token type, followed by one or more data elements. The order of these data elements is determined by the record type. Event type and tokens within the record characterize different audit records. Some tokens consist of a single data element. (The text token is such a case.) Other tokens contain several data elements, as in the case of the process token, which contains the audit userID, the real userID, and the effective userID. For user-level and kernel-level events, tokens specify the process performing the event, any objects on which the event is performed, and tokens associated with the objects (including owner or mode).

BSM User-Level Audit Record Content

As previously noted, user-level generated audit records are created by applications that operate outside the kernel. The records are sorted alphabetically by program. The description of each record includes the following items:

- Name of the program
- Manual page reference (if appropriate)
- Audit event number
- Audit event name
- Audit record structure

BSM Kernel-Level Audit Record Content

Kernel-level-generated audit records are created by system calls used by the kernel and are sorted alphabetically by system call. The description of each record includes:

- Name of the system call
- Manual page reference (if appropriate)
- Audit event number
- Audit event name
- Audit event class

- Mask for the event class
- Audit record structure

System Audit Management Tools

Solaris BSM provides integrated audit trail management commands that allow the auditor or system administrator to perform a variety of audit trail functions. `Auditreduce` allows the auditor to perform post-selection of events, keying on attributes such as time intervals, specific user identifiers, and specific event identifiers. `Praudit` translates the audit records from their native binary format into a user-selected format. Humans can read these formats, but they are not otherwise interpreted and can serve as input for very basic reporting mechanisms.[3]

3.2.5.2 Operating System Example 2: Windows NT

Microsoft Windows NT Server operating system provides data sources in the form of its event-logging mechanisms. These mechanisms consolidate the information gathered from operating system and other system sources.

NT Event Logging Mechanism Structure

Windows NT event-logging mechanisms collect three types of system events: operating system events, security events, and application events. Each type of event is recorded in a separate log.

The system log consists of events generated by the Windows NT operating system components. These events include such events as driver or other component failures, application software crashes, and errors associated with data loss. The event types recorded in the system log are predetermined by the Windows NT operating system.

The application log consists of events recorded by applications. For instance, a database program might send an information event to the application log when the program successfully completes a backup operation. Event types recorded in the application log are determined by the application developers, and software toolkits are provided to help them use this logging feature.

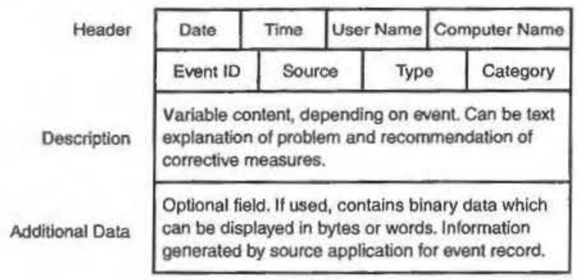
The security log consists of events that are defined as security-relevant. These events were derived from the TCSEC C2 definitions of auditable events. They include valid and invalid logins and logoffs, and events related to system resource use, especially those having to do with the creation, deletion, and alteration of system files and other objects. Unlike system and application logs, security logs are accessible only to system administrators.

Although all events are of interest to those attempting to reconstruct system activities, the security log events are the primary focus of intrusion detection systems.

Event Log and Record Formats

Event logs are composed of sets of *event records*. Each event record is divided into three functional parts: the header, a description of the event, and an optional additional data field. Figure 3.3 shows the structure of an event record. Security log entries usually consist of the header and a description of the event.

Figure 3.3 Format of Windows NT Event Record


Header
Date
Time
User Name
Computer Name
Event ID
Source
Type
Category
Description
Variable content, depending on event. Can be text explanation of problem and recommendation of corrective measures.
Additional Data
Optional field. If used, contains binary data which can be displayed in bytes or words. Information generated by source application for event record.

The event record header consists of the following fields:

- **Date** Identifies the date of the event.
- **Time** Identifies the time of the event.
- **User Name** Identifies on whose behalf the event occurred. This identifier can be the primary userID, a client ID, or both, depending on whether the Windows NT impersonation function is invoked. Impersonation happens when the operating system allows one process to inherit the security attributes of another. The security-event-logging mechanism reflects both the userID and the impersonation ID when impersonation has occurred.
- **Computer name** The name of the computer on which the event took place. This information simplifies the audit review when users centralize security administration functions across enterprises.
- **Event ID** A numerical identifier for the event type. This field is usually mapped to a text identifier (event name) in the description field of the event record.

- **Source** The software responsible for generating the event record. The source can be an application, a system service, or a device driver.
- **Type** An indicator of the event's severity. The available types depend on the type of log. In the system and application logs, the type can be error, warning, or information, in descending levels of severity. In the security log, the types can be success audit or failure audit.
- **Category** The triggering event type, used primarily in the security log to indicate the event type for which success or failure auditing has been enabled.

NT Event Log Management Features

Windows NT provides numerous features to allow system administrators to manage the operating system event log mechanisms. For instance, administrators can limit the size of the event logs and specify how to deal with the files as they approach this upper limit. The options include overwriting the oldest log records with new, in effect creating a circular queue; overwriting the oldest log records only if they are of a certain age; and halting the system until the event log file is manually cleared.

System and application event logging start automatically when the system is started. Logging stops when the log files are full and the system configuration specifies that they must be manually cleared. Security event logging, on the other hand, must be enabled by someone acting in an administrator capacity.

Security Logging and the Audit Policy

Security logging is governed by an audit policy, which is set up by using an auditing policy dialog box. The audit policy specifies the types of events to log and can be specified in terms of actions, users, and objects. The security event record shows the time and date of the action, the action performed, and the user responsible for performing it. Log entries can be generated for both successful and failed actions, recording evidence of actions that took place as well as attempts to perform actions that may have been prohibited by policy.

To audit specific files or folders, two separate steps are required. First, the audit policy is set to enable file and object auditing for the entire system. In TCSEC terms, this step renders these access events *auditable*. Next, the file property settings are accessed for each file, in order to turn on auditing for that file. This step corresponds to the TCSEC concept of making these events *audited*. Figure 3.4 shows the types of directory and file-access actions that can be audited under Windows NT, specifying permission settings that correspond to each event selection.

Figure 3.4 Audit Event Settings for Files and Directories

	File						Directory					
	Read	Write	Execute	Delete	Change permissions	Take ownership	Read	Write	Execute	Delete	Change permissions	Take ownership
Changing ownership						X						X
Changing permissions					X						X	
Changing Attributes		X						X				
Displaying owner and permissions	X	X	X				X	X	X			
Displaying Attributes												
Displaying names of files	X		X				X		X			
Displaying file data							X					
Altering contents	X											
Creating subdirectories and files												
Going to subdirectories								X				
Running file									X			
Deletion			X									

Tuning NT Audit

Balancing the performance loss associated with NT file audit with the benefit of additional detection capabilities still remains an art. Although you can simply set up audit for all system directories and subdirectories with a few keystrokes, this approach has the undesirable effect of slowing down system performance. On the other hand, auditing selectively at fine grain allows you to detect significant security problems (such as the execution of macro viruses and trojan horses). Performing this balancing act may be intuitively obvious only to the most experienced security administrators.

Audit-management features, which are provided as part of some commercial intrusion detection systems, can help system administrators fine-tune audit features, thus optimizing the audit to meet security and performance goals. This can be of great value to those charged with protecting an enterprise running Windows NT.[4]

3.2.6 Audit Reduction

Audit reduction is the process of filtering audit logs, identifying and removing information that is redundant or irrelevant. This process represents the classic problem of finding a needle in a haystack. Historically, there have been several significant impediments to designing and implementing a data reduction operation. Key to the reduction process is the capability to introduce some determinism to an inherently nondeterministic process. In other words, given the capability to state facts such as "Event X will always trigger events *Y*, *Z*, and *K* under conditions *A* and *B*," we can reduce the event stream consisting of (*X* [under conditions *A* and *B*] followed by *Y*, *Z*, *K*) to event *X*.

The problem with this approach is that (especially in multiprocessing, multitasking systems) this level of determinism is simply not present. Furthermore, the complexity of modern operating systems leads to scenarios in which a single high-level command triggers the generation of thousands of audit records. Worse yet, the order in which the records are recorded in the audit log is not consistent. For instance, a Sun OS UNIX `ls` command executed on a desktop workstation can generate more than 1,500 audit records. Repeating the command even seconds later often results in a different number of audit records, in a different order.

Research from the mid-1990s suggests that significant audit reduction can be done on records generated by trusted processes.

Note

In this context, *trusted processes* refer to those processes certified as supporting a security goal. Trusted systems most often refer to systems developed in compliance with the Department of Defense's TCSEC. In practical terms, trusted processes are those considered free of security problems.[4]

In this work, the assertion is made that when a user command spawns a process that is trusted (and that generates only trusted subprocesses), it is sufficient to record only the audit record corresponding to the user command, and to discard all the subsequent audit records for that process. When the process generated by the user command is untrusted, we record all the audit records for the untrusted processes and subprocesses spawned by the command. Again, if the untrusted process spawns trusted subprocesses, we can discard the event records associated with the trusted subprocesses. An event that fails is assumed to be a potential attempt to violate security protections. Thus, all records for the transaction from the time of failure until the completion of the transaction are recorded in chronological order.[5]

3.2.7 System Logs

A system log is a file that reflects various system events and settings. UNIX operating systems provide a rich assortment of system logs, along with a common service, **syslog**, which supports generating and updating event logs via the **syslogd** daemon. Although a rich lexicon of standard formats and definitions can be used in generating and interpreting syslog entries, the security of the logs so generated is considered to be weaker than that of kernel-generated operating system audit trails.

This perception of system logs as less trustworthy than operating system audit trails exists for several reasons. The log-generation software is usually running as an application and is thereby easier to subvert or otherwise modify than is the audit subsystem. Furthermore, the logs are usually stored in unprotected directories on the system. This convention makes it easy for an attacker to locate and destroy or alter the files. Finally, the logging operation is a simple text write, one line of input per log entry. Operating system audit logs are usually more cryptic, with many providing schemes for detecting alteration.

Several ways of mitigating some of the security problems associated with system logs have been suggested. For instance, Spafford and Garfinkel[6] propose that a secure logging host be established by connecting a dedicated microcomputer (perhaps one that has been retired from service due to obsolescence) to the host being monitored by using a serial connection. In this scheme, all system logs would be periodically redirected to this system.

3.2.7.1 Why Collect Data from System Logs?

Despite their weaker protection levels, system logs are usually easier to review than operating system audit logs. If generating operating system audit trails is not feasible in a particular situation or if intrusion detection tools are not available to interpret those operating system audit trails, system logs are a valuable source of information for system security managers. Furthermore, as discussed later in this chapter as well as in Chapter 9, if you need to use system information sources for evidence in a court of law, multiple independent data sources, all indicating that the same chain of events occurred, are considered much stronger evidence than a single source.

3.2.7.2 Typical Content of Logs

UNIX provides a rich variety of system logs that are suitable for security review. Table 3.1 lists the Sun Solaris system logs that are sometimes used for intrusion detection, the files to which they are written, and the information they contain.

Table 3.1 Sun Solaris System Logs

Log Name	Content	File Written/Used
pacct	Commands run by users plus resource usage	/var/adm/pacct
lastlog	Most recent successful/ unsuccessful login for each user	/var/adm/wtmp
loginlog	All login failures	/var/adm/acct/sum/loginlog
sulog	All use of su command	/var/adm/sulog
utmp(x)	Lists each user currently logged in; utmpx is a more current extended version of log	/var/adm/utmp(x)
wtmp(x)	Time-stamped list of all user logins/logouts and system startups and shutdowns; wtmpx is a more current extended version of log	/var/adm/wtmp(x)
nis.trans	List of all changes in NIS namespace	/var/nis/trans.log

In addition to these system logs, UNIX provides a `syslogd` daemon that logs system information in a special log file. In situations that call for a great deal of custom software to run in a critical installation, this gives the software developer options. By including logging commands in the custom software that write event records using `syslogd`, the developer can extend intrusion detection capabilities to nonstandard system processes.[3]

3.2.7.3 Consolidation of Log Information

For intrusion detection purposes, system logs can provide information that can be combined with the other system information sources to more accurately determine the events occurring on a system at a given time.

Multiple logging mechanisms can serve as a valuable "sanity check" to those interested in finding security problems on a system. Using information in system logs as additional views of the system activities increases the likelihood of discovering intruders. Discrepancies between events recorded from different vantage points help to identify situations in which an intruder was only partially successful in making the evidence of an attack.

One of the most widely publicized network security incidents of the 1980s occurred at Lawrence Berkeley Laboratories. A system administrator, Cliff Stoll, was assigned the task of resolving a $0.74 discrepancy between two accounting logs generated by separate logging mechanisms on the same computer system. Stoll's investigation led him to a foreign agent who had hacked LBL's systems in an attempt to gain access to sensitive information about critical weapons systems. Stoll's account of the investigation, *The Cuckoo's Egg*, remains a classic in computer and network security circles.[7] Several strategies are available for consolidating logs in a fashion suitable for use in intrusion detection. One that is still widely used involves storing the raw logs in a database and then constructing queries that present the data from a variety of perspectives. In some cases, when external evidence indicates that a problem took place during a particular time interval, it can be very helpful to simply isolate the logs that correspond to that interval. You can then sort them according to user, system object, and order of occurrence. Some commercial intrusion detection systems provide this capability.

In other cases, an administrator or intrusion detection system may be able to tell from operating system audit trails or network traffic that something is amiss, but may not be able to confirm or dispel that suspicion. In this situation, synchronizing events reflected in low-level kernel audit records with coarser-grained log events can sometimes help determine whether further investigation and/or responses are required. In Chapter 4, "Analysis Schemes," the discussion of the consolidation of multiple data sources will continue.

3.2.8 Applications Information

So far, we have focused on collecting data for intrusion detection at the system level. This approach reflects a rather traditional belief of computer security: The only trustworthy data is data collected in the bowels of the system, out of reach of all but the most expert intruders. Although this perception can still be true, as systems and operating systems evolve, so do security threats and protection models.

In a world in which the speed and complexity of systems seems to grow without bounds, those trying to protect their systems naturally look first for signs of trouble that are comprehensible to mere mortals. In modern systems, application logs often represent the only available user-level abstractions of system activity. As such, these logs may well be the only places in which administrative security policies can be traced and noncompliance with those policies shown.

Many members of the intrusion detection community assert that in the future all event information of interest will be generated at the application level. This effect is likely due to multiple factors. Perhaps the most powerful force driving monitoring toward the application level is the advent of object-oriented and distributed systems. We already note this trend in current versions of Microsoft Windows NT, as many events formerly recorded at the operating system event log level have migrated to application data stores.

Almost all commercial operating system audit mechanisms support the generation of application-level audit entries, but few include audit features in their own applications. Those that do compensate by automatically turning off operating system kernel audit for processes invoked by the application. This measure can be risky because it is difficult to guarantee that the application is completely trustworthy, but this feature makes audit logs somewhat easier to comprehend.[2]

3.2.8.1 Applications Data Sources Example 1: Database Systems

One example of an application environment in which the need for audit trails and intrusion detection is apparent is the database management system. In many large organizations, the most critical information resources are housed and accessed strictly via the database management system. For example, many health care institutions store patient records and other sensitive data on large database systems. Government agencies maintain large databases of tax and voter registration records.

Early in the Trusted Systems Initiative, the U.S. Department of Defense chartered research to explore the issues associated with application-level auditing of trusted database management systems. This research isolated several issues associated with audit and intrusion detection on these systems—some are similar to those challenges associated with operating system audit (for instance, performance issues); some are not.

Issues Associated with Data Volume

One such issue concerns audit data volume. In databases, as in operating systems, gigabytes of audit data can be generated in a matter of hours. This factor suggests several considerations when designing intrusion detection systems: suitable compression/archival techniques for audit data, policies and techniques for performing audit data reduction, and granularity of audit control. *Granularity of audit control* refers to whether the audit mechanism can be directed to record audit records for each event type; or whether the audit records must be switched on or off for an entire group of events, not just one. The latter approach is more apt to lead to problems with audit data volume because it forces the audit mechanism to record an entire group of events just to capture the needed one.

Temporal Issues

Another issue involves the time of audit data generation. Depending on where a software developer places a call to the audit subroutine to generate an audit event, a discrepancy can develop between a system event and the recording of an audit record reporting that event.

If you generate an audit record at the beginning of the execution of the transaction, you run the risk of having the time of execution off by fractions of a second to minutes or even hours, depending on the time it takes the execution of the transaction to complete. Worse yet, you also run the risk of the transaction crashing or aborting after the audit record is generated, resulting in a discrepancy between what the audit trail indicates happened and what actually happened.

If, on the other hand, you generate an audit record at the end of the execution of the transaction, you eliminate the possibility of directing the operating system or application to take corrective action in response to a detected problem. (An example is blocking access or commands upon evidence that something is amiss.)

Issues Associated with Level of Abstraction

Similar tradeoffs occur in both database systems and operating systems between the granularity of audit records and the ability of the user to make sense of the audit data. The lower the level of abstraction reflected by the data, the subtler the attack that the system can reveal. The higher the level of abstraction, the more intuitively obvious the reported events are to the reviewer. For this reason, in database systems (as in UNIX systems) many advocate the use of database transaction logs when performing intrusion detection. The use of these transaction logs and UNIX system logs present some of the same problems. For instance, they are not as well-protected as system audit trails are. If the database software itself is corrupted by an adversary, the transaction log can lie to you about what really happened, showing that a transaction occurred when operating system audit trails indicate that another operation was substituted for the purported transaction. Because many of the critical information resources that customers want to protect reside in large databases, database management system audit mechanisms comprise an important information source for intrusion detection systems. This remains an extremely important area of research.[8]

3.2.8.2 Applications Data Sources Example 2: WWW Servers

Because the current explosion in electronic commerce is built upon the World Wide Web, the Web server application is a common source of application-level information. Most Web servers support access log mechanisms that can be a rich source of information.

Access Log Formats

Current Web servers support at least two standard formats for log files. The first is the Common Log Format (CLF), derived from the early versions of the NCSA Web server.

(The National Center for Supercomputing Alliances was the source of the first widely deployed Web browser.) The second is an extended log format that depends on the specific Web server.

CLF logs contain the information displayed in Table 3.2.

Table 3.2 CLF Log

Field	Format
The host name of the visitor accessing the site	"Host.subnet.domain.net"
rfc931	information returned by `identd` for this user; otherwise "-"
The username if a userID was sent for authentication	userID
The date and time of the request plus time zone information	[DD/MMM/YYYY]:HH:MM:SS +TZO]
The name of the page requested and the protocol used by the server to communicate the page	"GET xxx.host.subnet.domain.net"
The status code for the request (200 indicates success)	NNN, "-" if not available
The number of bytes returned by the request	NNNNN, "-" if not available

A typical CLF access log entry would appear as follows:

```
duh.infidel.net-bbace[5/May/1999:02:00:03 +0600]"GET/~sret1/HTTP/1.0" 200 1893
```
[9]

An extended log file might include the data fields included in CLF plus additional information. This extended format is displayed in Table 3.3.

Table 3.3 Extended Common Log File Format Description

Field	Format
The host name of the computer accessing the site	"Host.subnet.domain.net"
rfc931	information returned by `identd` for this user; otherwise "-"
The username if a userID was sent for authentication	userID

continues

Table 3.3 Continued

Field	Format
The date and time of the request plus time zone information	[DD/MMM/YYYY]:HH:MM:SS +TZO]
The name of the page requested and the protocol used by the server to communicate the page	"GET xxx.host.subnet.domain.net"
The status code for the request (200 indicates success)	NNN, "-" if not available
The number of bytes returned by the request	NNNNN, "-" if not available
The address of the referring URL to this page (if visitor utilized) a hot link to access site	`http://www.refsite.com/pagedir/page`
The browser name and version used by the visitor	"browsername/versionnumber" (OS details)

Therefore, the preceding access results in this log entry:

```
duh.infidel.net-bbace[5/May/1999:02:00:03 +0600]"GET/-sret1/HTTP/1.0" 200 1893
"http://www.yahoo.com/security/anomalies" "Mozilla/2.0(X11;I;SUNOS A 06.07.8000/725)
```
[10]

On the Microsoft front, Internet Information Server offers logs in either the CLF or a customized extended format. In addition, the server offers the administrator the option to log entries to a text file or to an open database connectivity (ODBC) database for later queries.

Because Web server logs are used as a basis for measuring Web site statistics that affect revenue generation in commerce sites, we can expect Web server vendors to continue to provide both embedded application log generators and some modicum of protection to these logs.

3.2.8.3 Issues Associated with Application Audits

The issues associated with application audits mirror many of the fundamental challenges of intrusion detection. The first of these is the temporal ordering of audit events. Any time you attempt to make sense of a chain of events based on information collected from more than one point of view, you need a way of organizing the information. In intrusion detection, information is usually organized by recording or assigning a time to each event and then placing the event stream in order according to this time stamp. If the time stamp is missing, then you are limited to seeing things that occur in a particular instant of time.

The second class of issues concerning application audits involves combining the audit trails so that users can make sense of them. The term *composition* is sometimes used to describe the combining of data streams; when the event streams involve information at a higher level of abstraction, some people use the artificial intelligence term *fusion* for this process. Composition and fusion both present significant challenges for a number of reasons. We explore these reasons in more depth when we discuss current issues in Chapter 7.

3.2.9 Target-Based Monitoring

Target-based monitoring is a variation on standard host-based monitoring. The premise of target-based monitoring is that in systems in which resource constraints prevent comprehensive kernel-level auditing, the partial logging of system activities is still possible. To determine where to perform logging, we first assess the most critical or valuable objects within the system, and then construct monitoring mechanisms designed to collect stale information of the object. The state transitions of the object are compared to a security policy and any discrepancies recorded.

3.2.9.1 Definition of Target-Based Monitoring Approaches

The most common target-based monitoring approach uses cryptographic integrity checkers to monitor state changes in system objects, such as critical files. Although this approach to monitoring is static (unlike the dynamic approaches represented by audit and logging mechanisms), it is still a powerful addition to the intrusion detection arsenal. The difference between the static target-based approach and dynamic approaches can be described as follows: Target-based monitoring is analogous to snapshots; dynamic approaches are analogous to video images. Consider the difference between the two when they are used to record the security status of a file cabinet. Although the snapshot might not inform us immediately of a problem, if the camera is set to take snapshots at one-minute intervals, it can provide valuable information at a very low cost. Video monitoring, on the other hand, displays a problem as it happens. However, a video camera costs more than a film camera, and if no one is present to react to the information, the speed of detection may be irrelevant to the organization.

An integrity checker computes a cryptographic checksum for every guarded system object and stores the result in a protected location. The checker computes the checksum using a specialized algorithm called a *message digest algorithm.* (Such algorithms are also known as *cryptographic hash codes* or *hash functions.*) Message digest algorithms are designed with two goals in mind. The first goal is to make the algorithm "cryptographically strong," meaning that the possibility of the algorithm computing the same result for two different inputs is small enough to be considered nonexistent. The second goal is to have a small change in the input to the algorithm produce a large change in the output. Thus, even the tiniest change in the protected object will be apparent at the second stage of integrity checking, when the cryptographic checksum is recomputed and the result compared to the stored value.

3.2.9.2 Rationale for Target-Based Approaches

One justification for target-based monitoring approaches is the following: In UNIX-based operating systems, all items of interest to users—including network connections, devices, and processes—can be represented as files. These items are represented by structures called *inodes*. Figure 3.5 shows the structure of an inode.

Figure 3.5 Contents of a UNIX File System Inode[6]

Item	Location	Type	Size in Bytes
Time	Inode Modified *CTime*	Contents Modified *Mtime*	File Accessed *Atime*
	Reference Count	Location of Data (Disk Address)	

Integrity checkers were designed to augment a traditional feature in some file systems. The primary checksum mechanism provided in UNIX systems (sum) is a cyclic redundancy check (CRC). However, this checksum is generated by a well-known polynomial. Furthermore, because the original purpose of CRC polynomials was to detect errors in data communications over noisy channels, the polynomials were designed to detect random changes, not deliberate modification of file contents. Analysis of early hacker attacks discovered a common use of tools that allowed you to nullify the CRC's capability to detect changes in a modified file. Therefore, stronger mechanisms were clearly required.

The Tripwire® integrity assurance tool was developed by Gene Spafford and Gene Kim of the COAST project at Purdue University. Tripwire®[11] was written in the early 1990s to optimize the process of utilizing cryptographic hash codes to protect critical files and objects in UNIX systems. The strength of the cryptographic mechanisms and the numerous options allowed security personnel to focus on critical files, generating information that constitutes a powerful data source for any detection mechanism. The system is now available as a commercial product, with versions for Solaris, Windows NT, and Linux.[12]

3.3 Network-Based Information Sources

Network traffic is the most common information source in current commercial intrusion-detection products. In network-based approaches, information is collected from the network traffic stream as it travels on the network segment. In this section we consider network data as an information source for intrusion detection.

3.3.1 Why Network Sources?

Network-based information sources are popular for many reasons. Primary among them is the fact that the information gained by network monitoring is considered to come at low or no performance cost because the monitor simply reads packets as they cross its network segment. Therefore, running the monitor does not affect the performance of other systems on the network.

Another advantage offered by network-based information sources is that the monitor can be transparent to users on the network, thereby decreasing the likelihood that an adversary will be able to locate it and nullify its capabilities without significant effort. Because the primary resource required of the monitoring system is storage space, organizations can sometimes use older, slower systems to perform monitoring on a network segment.

Finally, network monitors can see evidence of certain classes of attacks that are not easily visible to host-based systems. These attacks include network attacks based on malformed packets and various denial-of-service attacks, including packet storms.

3.3.2 Network Packets

A network-based ID system views packet traffic on its network segment as a data source. This is usually accomplished by placing the network interface card in *promiscuous mode*. Promiscuous mode is a network interface setting that generates interrupts for all network traffic that crosses the segment. Although simple and powerful, this approach does not always work on modern network systems. For instance, in the case of switched networks the network switch acts to isolate network connections between hosts so that a host can see only the traffic that is addressed to it. Also, data that travels via other communications media (such as dial-up phone lines) cannot be monitored using this approach.

An interesting approach to providing network-based monitoring and data collection is provided by Network Flight Recorder (NFR). Developed by Marcus Ranum, whose prior accomplishments included the design and development of several of the earliest network firewalls, NFR is not actually a network-based intrusion detection tool, but a generic network monitor with APIs suitable for supporting add-on intrusion analyzers. The packet capture and reassembly features of NFR are a real boon to customers who want to build customized, network-based, intrusion-detection systems.

3.3.3 TCP/IP Networks

The Internet and the Transmission Control Protocol/Internet Protocol (TCP/IP) protocol that enables it represent the staging area for much of the behavior that is of security interest today. As the protocol standards dictate the steps required to use network packet traffic as a data source, it is helpful to understand how TCP/IP works, and how it structures and packages data.

3.3.3.1 A Brief History of TCP/IP

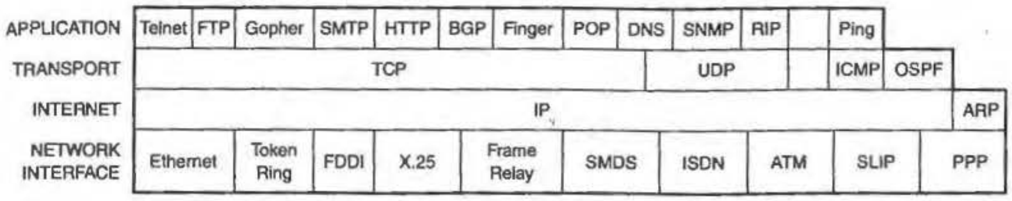
TCP/IP was created in the 1960s under the ARPANET initiative. Funded by the (Defense) Advanced Research Products Agency (ARPA), the goal of TCP/IP was to enable the military to use computer networks that were robust enough to withstand any kind of disruption, up to and including nuclear war. TCP/IP networks are packet-based, shared communications networks in which communications between systems on the network take place as sequences of discrete packets. The packets travel a series of segments, interconnected by devices called gateways and routers, which are designed to make intelligent decisions about the optimal paths that packets should take en route to their destinations.

3.3.3.2 Structure of the Protocol Stack

The TCP/IP stack contains four protocol layers, which are pictured in Figure 3.6. The four layers are stacked so that each one uses the services of the layer below it. They are organized this way:

- **Applications** Applications such as mail, video server, login, and file transfer are on top.
- **Transport** Next is a protocol layer, such as TCP, which supports the applications by providing a reliable "virtual circuit" between the endpoints of the network connection. The overhead associated with this function includes splitting the data flow into packets, performing error correction, and calculating sequence numbers in order to reconstruct packets in proper order at the destination.
- **Internet** Next is the IP protocol, which serves as a packet multiplexer. It affixes an IP header to each packet from the higher-level protocols, and then sends it to the appropriate device driver for transmission to the specified destination. The nature of IP layer service is that each packet stands alone; this is a datagram service, not a virtual circuit.
- **Network interface** Finally, the bottom layer consists of device drivers that manage the physical communications medium, such as an ethernet local area network or Fiber Distributed Data Interface (FDDI).

Figure 3.6 Simplified TCP/IP Protocol Stack[13]



3.3.3.3 IP Address Structure

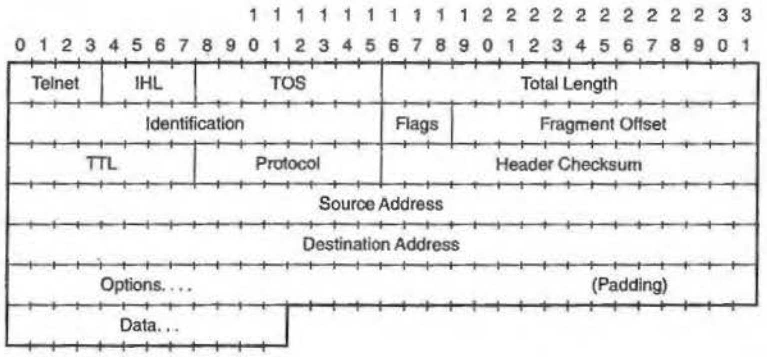
In TCP/IP, IP addresses are constructed of a 32-bit number that is divided into two parts: a network portion and a host portion. The host portion of the address is usually divided into a subnet and host address, where subnets are used to perform routing internal to an organization.

The number of bits used for the network is variable; many environments divide a single Class B network into 254 subnets. IP addresses are also characterized as a set of four octets. In the address, each octet is delimited by a period, yielding an address that looks like this: 125.30.254.1. Another scheme for addressing hosts on the Internet is to use a specialized distributed database called the Domain Name System (DNS). DNS translates the IP address into a textual address (for example, `fw3.infidel.net`).

3.3.3.4 Packet Structures

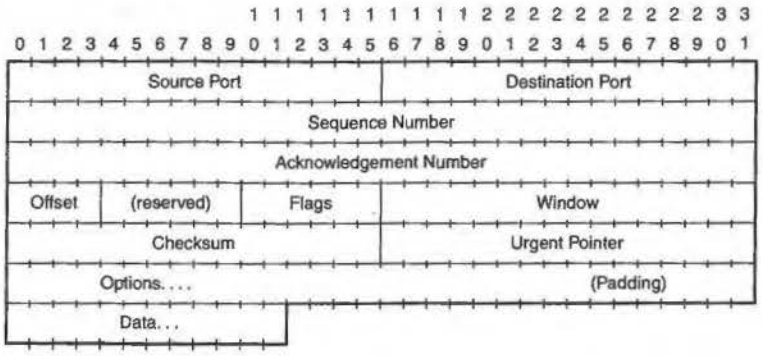
The TCP layer adds header information to packets, as shown in Figure 3.7.

Figure 3.7 IP Packet (Datagram) Header Format[13]



The packet with TCP header information is then forwarded to the Internet layer, where the IP datagram header is attached. The content of IP packet headers is shown in Figure 3.8.

Figure 3.8 TCP Segment Form[13]



Finally, the packet is passed to the network interface and sent on to the intended destination, where the TCP layer strips off the header information and reconstructs the data stream from the source machine.

3.3.4 Packet Capture

Given this background information about the protocol stack and packet structures, we next turn to the following question: How do you actually grab packets from the network? We consider the two prevailing environments in which this step is performed, UNIX and Windows, and attempt to highlight the options available in each.

3.3.4.1 Windows/NT Packet Capture Options

Prior to the release of Windows NT, many of the packet-capture options available for Microsoft platforms were in the form of tools that captured raw data from ethernet network adapters. Examples of these tools included Gobbler, Ethdump, and Ethload.

Starting with early versions of Microsoft Systems Management Server (SMS), Microsoft provided the Microsoft Network Monitor, a packet sniffer designed to run under Windows NT. Effective with the release of Windows NT Server 4.0, Microsoft fielded the Network Monitor Tool as a standard feature in the operating system. It offered ethernet frame capture with a variety of protocol parsers, a filter language, and a Windows-standard user interface.

Microsoft also provides an interface that allows a remote machine to connect and capture local data. This password-protected feature has generated some interest within the security community because at least one hacker group has broken the authentication mechanism!

3.3.4.2 UNIX Packet Capture Options

The body of packet capture utilities in UNIX is far richer than in Windows environments. One reason is that most network technologies have evolved within the UNIX environment, and UNIX's popularity has in no small part been driven by this support of networking. In general, Windows also appears to have been developed with the assumption that most administrators in Windows environments have little to no interest in tinkering with the low-level operating system code.

That observation aside, let's take a look at the approaches to network packet capture under UNIX. This survey starts "in the olden days" to accommodate the fact that many users have not, for various reasons, chosen to run the most current operating system versions. This approach is also helpful for understanding how changes in the low-level packet-handling schemes have driven requirements for packet capture and analysis.

Early Packet Capture Approaches

The requirement for packet capture facilities in UNIX is rooted in the need for network managers and engineers to have diagnostic and analysis tools. As enterprises have become more reliant on networked systems, the importance of isolation and correction of network problems has grown.

The earliest work in the packet capture and filtering area resulted in the Xerox Alto packet filter. This packet filter was adapted to UNIX by a team from CMU and Stanford in the late 1970s/early 1980s. The CMU/Stanford Packet Filter (CSPF) was designed to run on a DEC PDP-11 computer. The CSPF design was carried forward into standard packet filter mechanisms that are included in more modern UNIX operating systems: NIT (SunOS), UPF (Ultrix), and Snoop (IRIX).

The CSPF used a memory-stack-based filter mechanism, optimized to the older system architectures that were common at the time. It used an expression tree to perform filtering operations, which led to some redundant computations and subsequent losses in performance. The package was restricted to parsing fixed-length packet headers and was furthermore restricted to 16-bit data types (a problem, considering that Internet addresses and TCP sequence numbers are 32-bit data!). Despite its problems, CSPF had significant strengths as well, notably a packet-handling scheme that allowed CSPF to be protocol independent and an abstraction scheme that allowed users to more easily describe and implement the filtering mechanism.

Berkeley Packet Filter

Realizing that there was a mismatch between the CSPF design and more modern RISC-based CPUs, researchers from Lawrence Berkeley Laboratory developed a new kernel architecture for packet capture in the early 1990s, called Berkeley Packet Filter (BPF). BPF implemented two architectural improvements over CSPF. First, BPF switched from a memory-stack-based filter to a register-based filter (a better match with the register-based RISC processors prevalent in modern systems); second, it used the massive increases in memory address space by relying on a nonshared memory model. The performance improvement over CSPF-based mechanisms was impressive, showing up to 150 times greater performance tested against the same hardware and traffic combinations.

BPF has two primary components: a network tap and a packet filter. The network tap takes packet information from network device drivers and carries it to listening applications. After the packet reaches the listening applications, the filter grabs the data and, based on the filtering operation, decides whether the data should be delivered to the listening application. If so, it decides how many bytes of data should be passed. (For instance, a return value of zero indicates that the packet should be completely dropped.)

Two widely used network-monitoring applications use BPF, tcpdump, and arpwatch. Tcpdump is a network monitoring and data acquisition tool that performs filter translation, packet acquisition, and packet display. The filter translation function provides a high-level language for characterizing filters, along with a user-transparent compiler and optimizer that yields BPF programs. Tcpdump is available in versions for SunOS, AIX, Ultrix, and BSD versions of UNIX. Arpwatch is a monitoring program that tracks ethernet to IP address mappings, and notifies administrators when new bindings are made or abnormal activity is observed.[14]

3.3.4.3 Libpcap

Some intrusion detection vendors (notably Network Flight Recorder) use libpcap, the packet-capture library used by tcpdump. Libpcap is a system-independent interface that provides a portable framework for low-level network monitoring. The portability across system platforms provided by libpcap is of considerable value to those who function in heterogeneous system environments. Other advantages associated with the use of libpcap include the capability to download any first-cut packet filtering into kernel memory, which presents significant improvement in performance. In addition, libpcap is supported in Linux, which is growing in popularity as a platform for network monitoring and intrusion detection applications.[15]

3.3.4.4 STREAMS-Based Packet Capture

STREAMS is a system programming model for writing device drivers. Originating in AT&T versions of UNIX and supported in a variety of major UNIX operating systems (including Sun Solaris, HPUX, SCO UNIX, and IBM's AIX), STREAMS provides an extensive collection of system calls, kernel resources, and kernel utility routines that create, use, and dismantle a data stream. A *stream* is defined as a full-duplex processing and data transfer path between a kernel-resident driver and a process in user space. A stream is designed as a pipe structure, with data passing between a driver and the stream head via messages ("upstream" or "downstream," depending on whether the messages are traveling from the driver to the stream head or vice versa).

Packet capture and buffering is offered as part of the standard STREAMS libraries (pfmod and bufmod, respectively, in Solaris), although the architecture of STREAMS makes the performance suboptimal when compared to the BPF-based approaches. The data link provider interface (DLPI), a sniffing interface, is also widely used on systems that use STREAMS.[16]

3.3.5 Network Devices

In addition to packet sniffers, various other network devices can yield information that affects the detection of problems. For example, a network management system can provide performance and utilization statistics that are extremely helpful for determining whether a detected problem is likely to be security-related or due to other system factors. Because an important design goal is to perform monitoring and detection in a fashion that optimizes the performance of the system being protected, investigating and using existing information sources is always preferable to custom building or duplicating the data-collection process.

3.3.6 Out-of-Band Information Sources

A category of data sources that is often neglected by those researching or designing intrusion detection systems is "out-of-band" sources. This category encompasses information that comes to the system by nonsystem, sometimes manual means. Although the notion of a human operator assisting and directing the system as it analyzes information for signs of problems is not as glamorous as a sophisticated artificial intelligence approach, relying on human assistance is still an effective way to perform this function.

Human input can be in the form of information that is manually generated, recorded, or reported. An example of this type of input is the manual logging of events that occur in the system environment. These manual logs can include entries documenting hardware failures, power interruption, failures of systems not within site control (such as telephone service), or anomalous events (threats or natural disasters) that might affect the outcome of an event analysis.

An additional human input comes in the form of user interaction with intrusion detection control components. This type of input can consist of system configuration or policy entry at the time of system installation and setup. It can also take the form of active interaction with the system console, responding to detected problems by directing the system to take additional action. Several generations of intrusion detection systems are likely to be required before detection schemes are mature and reliable enough to allow unsupervised operation. For some environments, the total automation of intrusion detection analysis functions may never be appropriate. In the meantime, you should consider intrusion detection systems as diagnostic tools that allow you to "see" system activity through a security-savvy prism, using this higher-level view to gain additional expertise in recognizing and dealing with security problems.

3.4 Information from Other Security Products

In general, the more event information that is considered in performing intrusion detection analysis, the more accurate and sensitive the results. This relationship is especially true in performing intrusion detection on networks, where stand-alone security products are common.

Many firewalls, I&A systems, access control systems, and other security devices and subsystems generate their own activity logs. These logs contain information that is, by definition, of security significance; they are therefore of particular value to the intrusion detection process. Including these logs as information sources is an obvious way to improve the quality of the intrusion detection process.

The process of integrating and analyzing event logs from other components of the system security infrastructure represents a significant and enduring role that intrusion detection systems can play. This role continues to be relevant, even as strong encryption or other measures address those threats we consider of greatest concern today. Although technical environments change, the attack strategies of adversaries remain relatively stable. Because an elementary step in attacking a system is to locate, investigate, and then nullify the existing system protectious, monitoring security products will remain a stable requirement.

3.4.1 An Example of a Security Product Data Source

Table 3.4 presents a format diagram of the firewall log file generated by Firewall-1, a CheckPoint Technologies product. It allows you to see the transactions processed by the firewall, including mappings of ostensible sources and destinations of connections, the names of system objects associated with the transactions, and other information that was selected for inclusion.

Table 3.4 Firewall-1 Log File[17]

Field No.	Name of Field	Contents
1	Number	Transaction identifier
2	Date	Date of event—D(D)MMMYY
3	Time	Time of event—HH:MM:SS
4	Action	Accept or deny
5	Type	
6	Origination	
7	Alert	
8	Interface name	MAC address of ethernet card
9	Interface direction	Inbound/outbound
10	Protocol type	TCP or UDP
11	Src host	IP address of source
12	Dst host	IP address of destination
13	Type of service	Network service type
14	Port number for the source	Port number addressed by packet
15	"Rule"	Firewall rule tripped
16	Elapsed time	Time since session started
17	Packets for this session	Number of packets associated with session
18	Number of bytes	
19	Authenticated username	
20	Messages	

3.4.2 *Organization of Information Prior to Analysis*

As we noted when we discussed application-level data sources, issues arise when integrating data sources coming from different points in the network, especially when the sources are collected from different levels of abstraction. Security product data sources are no different in this regard.

Perhaps the best rule of thumb to apply in this case is to first temporally order the data according to a standard time reference. Network Time Protocol can be helpful in this regard; so can other forms of trusted network time service. After the data is sorted in chronological order, correlate log events to threat scenarios of interest. Alternatively, correlate log events to security policy noncompliance. Because security products are often the first heralds of attack, log events that correspond to attacks targeting the security products are usually reliable attack signatures in themselves.

As before, this binding of events to a time source, followed by correlation to a threat model, can be done either manually or with the assistance of a system. No system currently performs this correlation with enough reliability to eliminate the "man in the loop." For certain systems and environments, a human will remain a part of the process for some time to come.

3.4.3 Other System Components as Data Sources

Other system components also provide information that can be of value in intrusion detection. Many of these components are not often considered as legitimate data sources because they are not considered to be legitimate parts of the systems. Let's consider an example.

Early intrusion detection systems relied on a great many rules and detection algorithms to detect *masqueraders,* intruders who gained access to systems by using userIDs and passwords purloined from legitimate users. A variety of system audit trail approaches, ranging from statistical profiling of user activity (establishing patterns of "normal" behavior) to extensive rule trees (outlining the specific behaviors that a masquerader might exhibit) were proposed.

These approaches had a high false-alarm rate, and the amount of time devoted to investigating alleged masquerader attacks was appreciable.

In this case, information from physical access control systems can help determine whether a questionable access to the computer system is an attack. In this situation, if the intrusion detection system detects a suspected masquerade on the internal network, it can then query the physical access control system. If the physical access system indicates that the user in question is not on the premises, the implication is that the diagnosis of a masquerader attack is correct.

In another example of an out-of-band information source, consider a case in which a hacker attacks the network from a dial-up connection. After the specific modem used by the hacker is identified, the telephone switch caller ID channel might provide information that is useful to investigators who hope to identify the hacker. Telephone system traps and traces have also been used for this purpose.

3.5 Conclusion

In this chapter, we took a look at the plethora of data sources available for intrusion detection. We considered sources from different points within the system at different levels of detail and different levels of abstraction. In the next chapter, we begin to analyze the information drawn from these sources.

Endnotes

1. National Computer Security Center. "Department of Defense Trusted Computer System Evaluation Criteria." Orange Book, DOD 5200.28-std, December 1985.
2. Price, K. E. "Host-Based Misuse Detection and Conventional Operating Systems' Audit Data Collection." Master thesis, Purdue University, December 1997.
3. *Solaris Reference Manual Answerbook.* Sun Microsystems, Inc. Mountain View, CA, 1994–1998.
4. *Windows NT Reference Manual.* Microsoft Technet, Microsoft Corporation, Redmond, WA, 1999.
5. Lichtman, Z. L. and J. F. Kimmins. "An Audit Trail Reduction Paradigm Based on Trusted Processes." Proceedings of the National Computer Security Conference, October 1990.
6. Garfinkel, S. and E. H. Spafford. *Practical UNIX and Internet Security, Second Edition.* O'Reilly and Associates, 1996: 290.
7. Stoll, Clifford. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage.* New York, NY: Doubleday, 1989.
8. Schaefer, M., B. Hubbard, D. Sterne, T. K. Haley, J. N. McAuliffe, and D. Woolcott. "Auditing A Relevant Contribution to Trusted Database Management Systems." Proceedings of the Fifth Annual Computer Security Applications Conference, Tucson, AZ, December 1989.
9. World Wide Web Consortium. "Logging Control in W3C httpd." in httpd documentation: User's Guide `http://www.w3c.org/daemon/user/logging.html`.
10. Hallam-Baker, Phillip and B. Behlendorf. "Extended Log File Format." World Wide Web Consortium Working Draft, WD-logfile 960523, *WWW Journal,* issue 3, March 1996.
11. Tripwire® is a registered trademark of the Purdue Research Foundation.
12. Kim, G. H. and E. H. Spafford. "Tripwire: A Case Study in Integrity Monitoring." *Internet Besieged: Countering Cyberspace Scofflaws,* edited by Dorothy and Peter Denning, Addison-Wesley, 1997.
13. Kessler Hill Associates, Inc. C. Gary. "An Overview of TCP/IP Protocols and the Internet." April 23, 1999: `http://www.hill.com/library/tcpip.html`.
14. McCanne, S. and V. Jacobson. *The BSD Packet Filter: A New Architecture for User-Level Packet Capture.* 1993 Winter USENIX Conference, San Diego, CA, January 1993.

15. Ranum, M.J., K. Landfield, M. Stolarchuk, M. Sienkiewicz, A. Lambeth, and E. Wall. *Implementing a Generalized Tool for Network Monitoring.* Proceedings of the Eleventh Systems Administration Conference (LISA '97), San Diego, CA, October 1997.

16. *Streams Driver Overview.* Driver Development for SCO Systems, The Santa Cruz Operation, Santa Cruz, CA, 1999.

17. Sundstrom, Peter. "Firewalls-1 Report Summariser (fwlogsum version 3.6.0)." www.ginini.com.au/tools/fw1, August 1999.

CHAPTER 4

Analysis Schemes

"Is there any other point to which you would wish to draw my attention?"
"To the curious incident of the dog in the nighttime."
"The dog did nothing in the nighttime."
"That was the curious incident," remarked Sherlock Holmes.
- *Sir Arthur Conan Doyle*

Given a variety of rich information sources about the system activities you're charged with monitoring, the next stop on the intrusion detection circuit is analysis. With analysis, you're faced with the core issue of intrusion detection: What is going on here, and should I be interested in or concerned about it?

Within the analysis function, information is synchronized, classified, and subjected to scrutiny of various types to identify activity patterns of security significance. This chapter covers the vast area of analysis, considers the various goals of analysis, and discusses the issues you must take into account when devising an analysis scheme for a particular goal or system. It also explores the variables that affect the quality of the analysis.

4.1 Thinking About Intrusions

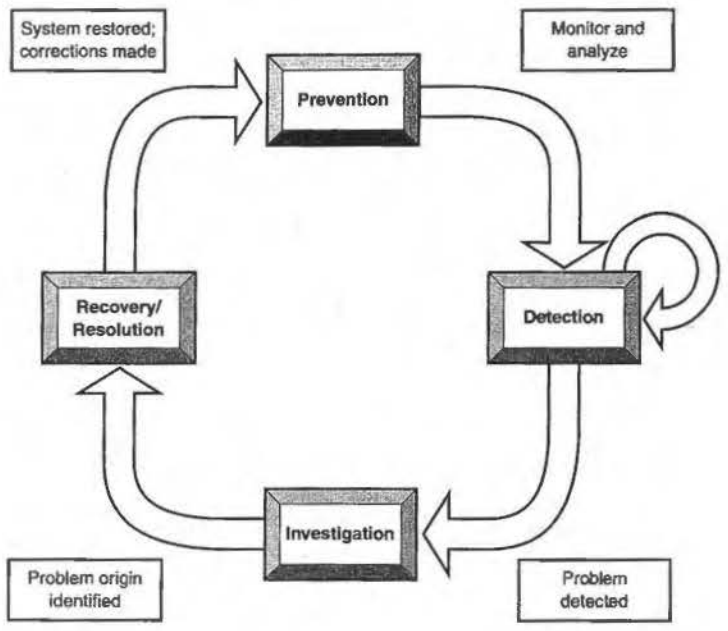
In considering the analysis functions of intrusion detection, let's start by articulating the detection process in intuitive terms. This definition will lend structure to the subsequent discussion of the fundamental issues and functions of analysis.

4.1.1 Defining Analysis

Analysis, in the context of intrusion detection, is organizing and characterizing data about user and system activity to identify activity of interest. This activity can be isolated either as it happens or after the fact. In some cases, further analysis is needed to establish accountability for the activity. Depending on the type of analysis, additional evaluation may be performed to tune or otherwise improve the outcome of subsequent analysis.

Intrusion detection is the second step in a general system security management process model, pictured in Figure 4.1. This model is helpful because certain analysis functions serve the needs of the investigative and resolution steps of the management process.

Figure 4.1 A General Model of Security Management



4.1.2 Goals

Security managers hope to gain several benefits by performing intrusion detection analysis. The overarching assumption is that people use intrusion detection systems to improve the security of their information systems.

- **Significant deterrence** One benefit a security manager seeks is to deter problem behaviors. The knowledge that analysis is being performed and the credible threat of discovery and punishment serve as a deterrent to intruders.
- **Quality control for security design and administration** Problems that are discovered in the course of analysis can indicate flaws in the design and management of

security for the system. The security manager, so notified, can then correct these problems before an adversary utilizes them to damage or steal information.

- **Useful information on actual intrusions** This information should be relevant, detailed, and trustworthy so that it can support criminal or civil legal remedies. It can also be used to identify needed corrections in the organization's security configuration, policy, or strategy.

4.1.3 *Supporting Goals*

Now that the goals of the analysis process are outlined, it's time to consider how each one drives specific functional requirements for an intrusion detection system. Structuring the requirements as sets of prioritized goals and subgoals helps to optimize the system.

4.1.3.1 *Requirements*

Intrusion detection analysis supports two basic requirements. One is, as we've mentioned before, *accountability*, which refers to the ability to link an activity with the person or entity responsible for it. Accountability requires you to be able to consistently and reliably identify and authenticate each user of the system. Furthermore, you must also be able to reliably associate each user with the audit or other event records of her activities.

Although the concepts of accountability are straightforward and common in business environments, they are difficult to implement in network environments. Unless you're in an environment featuring a single *sign-on system*, which centralizes identification and authentication for a large network, a user may have different user identities on different systems within the network. Because host-level audit trails reflect user activity in terms of the user's local identity, keeping track of user identity in activities that involve multiple hosts and user accounts requires additional processing.

The second requirement for intrusion detection analysis is *real-time detection and response*. This requirement includes the ability to quickly recognize the chain of events associated with an attack, and the ability to then block the attack (for instance, by terminating the network connection used by the attacker) or to shield the system from the impact of the attack (for instance, by tracing the commands sent by the attacker and restoring any altered files or objects to their pre-attack state).

4.1.3.2 *Subgoals*

Analysis also has subgoals. For instance, you might need to retain information in a form that supports system and network forensic analysis. Another subgoal might involve retaining some knowledge about the system performance or identifying problems that affect system performance. Yet another might involve archiving and protecting the integrity of event logs required by regulators or law enforcement entities.

4.1.3.3 Prioritizing Goals

After goals and requirements are articulated, they should be prioritized. Prioritization is necessary to determine the structure of the analysis subsystem. The priorities can be ranked by schedule (for instance, "this requirement will be serviced before that one"), by system (for example, "all requirements associated with system X will be serviced before those associated with other systems"), or by other attributes.

4.1.3.4 Trade-offs

At times, the goals and requirements of analysis may conflict. For example, one goal of analysis is to minimize the impact of analysis on the performance and resource consumption of the target system. However, the requirement to maintain logs for legal purposes often conflicts with this goal. In another example, the system overhead required to maintain accountability may affect the analyzer's ability to identify intrusions quickly enough to support active response.

4.1.4 Detecting Intrusions

To understand the nature of intrusion detection analysis, you should consider the full range of intrusion analysis available to customers, starting with those used in older systems. Although the focus here is on techniques suitable for automation, consider the full spectrum of discovery techniques for security intrusions and other violations of policy.

4.1.4.1 The Human Detector

Humans ("wetware" or "carbon units") are the most obvious and traditionally the most common source of intrusion information. A system manager may report that a machine is acting funny, or a user may report finding evidence of unauthorized activity on a system. Unfortunately, this approach is of extremely limited value because most incidents go unnoticed at sites where no intrnsion detection systems are in place. For instance, one early study showed that over a period of time in which a prototype intrusion detection system detected several hundred intrusions, only 2% of those were detected by the humans charged with performing manual intrusion detection.

4.1.4.2 External Events

Events external to the computer system often trigger suspicion of potential intrusion. These events include hiring and firing (especially of key personnel), reports of anomalies (such as sudden unexplained affluence of employees who work on critical systems), results of security penetration tests, and discovery of missing goods or information. These events are sometimes tracked as part of a traditional corporate fraud detection or loss control program.

News coverage or warnings posted to the Internet about particular classes of attacks also trigger suspicion of potential intrusion. This information appears to encourage the installation or use of antivirus tools.

4.1.4.3 Precursors to Intrusion

Next are the signs of intrusion that come from the victim systems. The first sign is evidence of an intruder having primed the system for future attack, which includes signs of trojan horse placement (that is, tampered system files) and unauthorized account additions to password files or trusted host configuration files (such as /etc/.rhosts in UNIX systems). These examples were the first symptoms of security problems targeted by early security tools such as COPS.

4.1.4.4 Artifacts of Intrusion

As precursors to intrusion alert customers to likely incidents, so do those indicators left in the wake of intrusions. These *artifacts*, evidence of past intrusions, can be reliable indicators of attack for both intrusion detection and vulnerability analysis. They can be discovered in real time (immediately upon the completion of the first step of an intrusion) or after the fact (during batch mode analysis of log files). Artifacts that are discovered immediately can be used by intrusion detection systems functioning in real time. Examples of artifacts of intrusions include password sniffer log files, unexplained system failures, and damaged files. Note that some artifacts are incidental to the intrusion (that is, they are not an intended goal of the intrusion) but excellent indicators, nevertheless, for detection purposes. Examples of such incidental outcomes include abnormal patterns of system resource use, discrepancies in accounting records (due to intruders deleting some, but not all, evidence of their activities), and unusual network traffic levels.

4.1.4.5 Observing Attack in Real Time

The final scenario for detecting misuse became possible with the advent and availability of intrusion detection systems—recognizing attacks by monitoring systems in real time. The ability to detect attacks by using these systems has opened the door to attack-blocking and other adaptive system responses to detected problems.

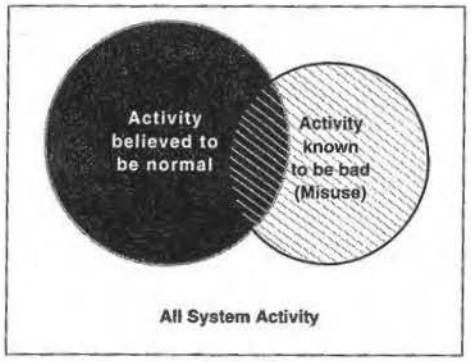
4.2 A Model for Intrusion Analysis

Analysis is the core function of intrusion detection. It can be as simple as an ad hoc decision table constructed by someone who has manually reviewed transaction logs, or as complex as a nonparametric system trained with millions of transactions.

We can look at the process of intrusion detection analysis from several perspectives. In this section, I define a model that covers all approaches to finding evidence of intrusions in system event records. It divides the process of intrusion detection analysis into three phases: constructing the analyzer, actually performing analysis of live data, and feedback or refinement of the process. Each of the first two phases has three functions: data preprocessing, data classification (classifying data as either indicative of intrusion, indicative of no intrusion, or "not sure"), and postprocessing.

Figure 4.2 depicts a view of the activity space, reflecting the view of misuse detectors, anomaly detectors, and possible gaps between the two. The fundamental debate between proponents of anomaly detection and proponents of misuse detection centers on the overlap of the regions representing "normal," "misuse," and "not normal" activities. Denning's initial assertion was that the region of "misuse" activity falls far enough outside the region of "normal" activity to use normality measures as the basis for finding misuse. Therefore, her assertion is that the intersection between the two regions is minimal. Proponents of misuse detection assert that the intersection is quite large, to the point that given observed errors in characterizing "normal" activity, it is pointless to use anomaly detection as anything but a crude alarm flagging activities that require closer scrutiny. The debate will continue for awhile yet, and this remains an open research area.

Figure 4.2 Misuse versus Anomaly Detection


Activity believed to be normal
Activity known to be bad (Misuse)
All System Activity

4.2.1 *Constructing the Analyzer*

In the analysis model, the first phase is the construction of the analysis engine. The analysis engine performs the core functions of preprocessing, classification, and postprocessing. For the engine to function properly, regardless of analysis approach, it must be tailored to the environment in which it is to operate; hence this phase is necessary even in rudimentary systems when it is performed solely as part of system development.

4.2.1.1 Collect and/or Generate Event Information

The first step of constructing the analyzer is collecting event information. Depending on the analysis approach, this phase might involve collecting event information generated by a system functioning in an operational environment, or collecting event information in a laboratory environment. In some cases, the event information may be handcrafted by a developer working from a set of formal specifications.

- **Misuse detection** For misuse detection, this part of the process involves gathering information about intrusions, including information about vulnerabilities, attacks, threats, specific attack tools, and observed scenarios of interest. At this time, information is also gathered about typical discretionary policies, procedures, and practices so that the behavioral model can accommodate organization-specific policy violations as well as problems associated with external attack.
- **Anomaly detection** For anomaly detection, event information is collected from the live system itself or from a system designated as similar. This information is needed to build baseline profiles indicating "normal" user behavior.

4.2.1.2 Preprocess the Information

After event information is collected, it goes through a variety of transformations in preparation for use in the analysis engine. It may be placed into a *canonical* or common format. This format is usually specified as part of the analyzer design. In some systems, the data is also structured (by placing it in a database or indexing it), some feature selection is performed (that is, the most relevant pieces of the data are identified and set apart), or other processing is performed.

- **Misuse detection** In misuse detection, the preprocessing of data usually involves transforming the event information collected into some sort of common form that corresponds to the data to be filtered when the intrusion detection system is placed into operation. For instance, the attack symptoms and policy violation information may be converted into state transition–based signatures or some sort of production system rules. In the case of a network-based intrusion detection system, the packets may be cached and Transmission Control Protocol (TCP) sessions reconstructed.
- **Anomaly detection** In anomaly detection, event data may be translated into arrays or tables, with some categorical data (for instance a system name), converted to numeric form (such as an Internet Protocol [IP] address). As in misuse data, different information may be converted into some canonical format.

Note

Canonical formats were originally used to allow a single analysis engine to monitor multiple operating systems, each of which usually had its own native format for event data. Intrusion detection developers could then leverage a common analysis engine to analyze data from many different operating systems. Because this practice allowed the developers to target new operating systems by translating the new event data to the canonical format, significant economic benefits were associated with this design approach. The utility of canonical formats still applies to situations in which one wishes to perform common analysis across a heterogeneous operating system environment. Some intrusion detection experts assert that enough organizations have converged on a common operating system for their entire enterprise so that canonical formats are no longer warranted.

At least one operational system accumulates user session data and then condenses it into numeric profile vectors. This reduces the event data to a form that takes up less space in memory and in storage. It also allows aggregation of certain data fields so that the analysis engine can easily recognize certain user activity patterns.

4.2.1.3 Build a Behavior Classification Engine or Model

The system developer, working from the design specification, then constructs a data classifier. This classifier separates event data corresponding to intrusive behavior from event data that appears not to indicate intrusions. The composition of this engine depends on the analysis approach.

Misuse Detection

In misuse detection, the data classification engine is built on behaviors described in terms of rules or other pattern descriptors. These rules or descriptors can be divided into single part (sometimes called *atomic*) signatures or multipart (sometimes called *composite*) signatures. An example of an atomic signature is one that detects a badly formed IP packet (with inconsistencies between the packet content and headers). An example of a composite signature is one that detects a UNIX `sendmail` attack (where the attack is a series of carefully timed steps).

One structure for a misuse detector classification engine is a *production* or expert system. *Expert systems* consist of a *knowledge base* containing descriptions of suspicious behavior based on knowledge of past intrusions (for iustance, those gathered in the previous step of this process) and *rules* that allow matching to be done against the knowledge base. These rules are usually structured as if-then-else statements.

Another structure for a misuse detector classification engine is a pattern-matching engine that represents intrusions as attack signatures (patterns) to be matched against audit data.

Because many systems built on this model are extremely efficient and reliable, it represents the most common approach to intrusion detection in commercial products.

Anomaly Detection

In anomaly detection, the classification model usually consists of statistical profiles of user behavior over time. These profiles can also be used to characterize the behavior of system processes, an important consideration given the widespread use of automated attack scripts. These statistical profiles may be calculated with various algorithms, using schemes that make allowances for gradual change in user behavior patterns. The profiles may be amended on either a fixed or variable schedule.

An example of an anomaly detection-based classification model is the Intrusion Detection Expert System (IDES), an instantiation of Dorothy Denning's seminal intrusion detection system model.

IDES defines its behavior classification model in terms of *measures*, single aspects of a user or subject's behavior on the monitored system. Figure 4.3 shows the classification of IDES measures (first distinguishing ordinal measures from categorical measures and then binary from linear). *Ordinal* or *continuous* measures are expressed in terms of a numeric count or quantification of the measure. *Categorical* or *discrete* measures are expressed in terms of identity and of frequency of occurrence.

Categorical measures are divided into two additional classifications, *binary* and *linear*. *Binary categorical* measures are characterized in terms of whether the measure occurred or not (a true/false switch), whereas *linear categorical* measures are characterized in terms of a score indicating the number of times a particular behavior occurs (a counter). The majority of measures utilized by IDES fall into the linear categorical group.[1]

Figure 4.3 IDES Measure Categories and Examples

	Ordinal (Continuous)	Categorical (Discrete)
Binary	CPU time used Number of audit records produced	Whether a directory was used Whether a file was accessed Whether audit records indicated use for day/week/month
Linear	▨	# of times each command was used # of system-related errors # of login failures in last hour # of audit events recorded # of files modified

4.2.1.4 *Populate It with Event Data*

After the model is built, it is populated with the collected and preprocessed event data. This instantiation of the model constitutes the analysis engine for the target system.

- **Misuse detection** Misuse detectors are populated with the preprocessed event data or contents of an attack knowledge base, a collection of information about attacks expressed in terms meaningful to the analysis engine.
- **Anomaly detection** Anomaly detectors are populated by running them against the collected reference event data (*training sets*), allowing the system to calculate user profiles based on this data. The fact that the historical data used to populate the anomaly detector is devoid of intrusions is often assumed without any corroborating evidence. Finding "clean" training sets for anomaly detectors remains a major issue.

4.2.1.5 *Store the Populated Model in a Knowledge Base*

Regardless of approach, the populated model is then stored in a predefined location, ready for operational use. At this point, the populated model contains all of the criteria for analysis, and in fact comprises the actual core of the analysis engine.

4.2.2 *Performing Analysis*

The second of the three phases in the analyzer is the operational analysis of a live event stream. In this phase, the analyzer is applied to live data to spot intrusions and other activity of interest.

- **Input new event record** The first step of performing analysis is taking an event record as generated by one of the information sources. Such information sources might be network packet traces, operating system audit trails, or application log files, and it is assumed that they have not been compromised.
- **Preprocessing** As in the construction of the analyzer, some preprocessing of the event data may be performed. The exact nature of this preprocessing depends on the nature of the analysis. Examples include threading together various TCP messages into a higher-level abstraction (a "session") and structuring process identifiers from operating system audit trails into high-integrity process trees.
  - **Misuse detection** For misuse detectors, the event data is usually converted to some canonical form. This form corresponds to the structure of the attack signatures. In some approaches, the event data is aggregated (collected to make up some minimum interval of interest, such as a user session, a network connection, or other high-level event). In other approaches, the data is *reduced* by combining some attrib-

utes, eliminating others entirely, and performing calculations on others to create new, more compact data records.

- **Anomaly detection** In anomaly detection the event data is usually reduced to a profile vector with behavior attributes expressed as scores and flags.

- **Compare the event record to the knowledge base** The formatted event record is compared to the contents of the knowledge base. The next step of the process depends on the results of this comparison and on the analysis scheme in question. If the record indicates an intrusion, it may be logged. If the record does not indicate an intrusion, the analyzer simply accepts the next event record and repeats the formatting and comparison.
  - **Misuse detection** In the misuse detector, the preprocessed event record is submitted to a pattern-matching engine. If the pattern matcher finds a match between an attack signature and the event data, it returns an alert. In some misuse detectors, if a partial match is found (if a pattern indicating a possible preamble to attack is matched), that fact may be recorded or the record cached in memory, awaiting further event information that can be appended to it to make a more definitive decision. Detection engines that can "remember" sequences of events are called *stateful* detectors.
  - **Anomaly detection** In anomaly detection, the contents of the user behavior profile for the session are compared to the historical profile for that user. Depending on the analysis scheme, a judgment is made as to whether the user behavior is close enough to the historical profile to be considered "normal" and therefore not indicative of attack. If the user behavior is determined to be abnormal, an alert is returned. Many anomaly detection–based intrusion detection engines also perform misuse-detection in parallel with this process, so some cross-pollination may occur between these different analysis schemes.
- **Generate a response** If the event record corresponds to an intrusion or other behavior of interest, a response is returned. Again, the nature of the response depends on the specific nature of the analysis approach. The response can be an alarm, a log entry, an automated response, or some other action specified by the operator of the intrusion detection system.

4.2.3 Feedback and Refinement

Certain analysis approaches have a third stage of analysis, one that runs in parallel with the main analysis function. The functions associated with this third phase are either maintenance of the analysis engine or other adaptive functions (refinement of rule sets or other system attributes).

4.2.3.1 Misuse Detection

In misuse detection systems, the primary activity that occurs in this stage is the update of signature databases to reflect information regarding new attacks. Given the almost daily reports of new attacks, this function is important. Many optimized signature engines accommodate this update activity "on the fly." (In other words, the system operator can update the signature database while the system is monitoring event data without interrupting the analysis process.)

Some misuse detection engines use this phase to combine and otherwise optimize the state-retention portions of the system, periodically eliminating records that have not been resolved.

Most misuse-detection-based analysis schemes have some notion of a maximum time interval over which matching of an attack signature occurs. This interval is known as an *event horizon*. For some, the event horizon might be from user login to logout (a session). Other schemes make this determination in terms of elapsed time since login or the last recorded event for that user.

In any event, because a significant amount of memory is necessary to retain this state information (especially when multiplied by the number of users, processes, and network connections active on busy systems), aggressive management of this state information is critical to the stability of a system. You see this effect in network-based intrusion detection systems in which "up the stack" reassembly of TCP sessions is performed.

In early misuse-detection efforts, the incomplete session records used to maintain state were labeled *orphans*. Performing the memory management was called (somewhat tongue-in-cheek) *running the orphanage* and was a frequent topic of discussion in some research and development circles.[2]

4.2.3.2 Anomaly Detection

Depending on the type of anomaly detection performed, the historical statistical profiles are updated periodically. For example, in IDES, the first statistical intrusion detection system, the profiles were updated daily. At that time, the summary statistics for each user were added to the knowledge base, and the oldest day's statistics (30 days old) were deleted.

The rest of the statistics (for day 1 through day 29) were multiplied by an aging factor. In this way, more recent behaviors had more effect on the determination of normal activity than older ones had. This practice is motivated by the need to accommodate changes in user behaviors associated with work schedules (for instance, accounting duties that may change over the course of a month) and user learning curves.[1]

4.3 Techniques

Now that I have explained the hows and whys of analysis and have outlined a general process for performing analysis, it's time to outline the specific approaches. As mentioned in Chapter 2, "Concepts and Definitions," analysis is divided into two parts.

Intrusion detection is composed of *misuse detection*, which searches event data for predefined patterns of misuse, and *anomaly detection*, which characterizes data in mathematical terms, searching event data for patterns of abnormality.

4.3.1 Misuse Detection

Misuse detection asks the following question about system events: Is this activity bad? Misuse detection involves encoding information about specific behaviors known to be indicators of intrusion and then filtering event data for these indicators.

To perform misuse detection, you need the following:

- A good understanding of what constitutes a misuse behavior
- A reliable record of user activity
- A reliable technique for analyzing that record of activity

In a nutshell, misuse detection is best suited for reliably detecting known use patterns. It suffers the deficiency that you can detect only what you know about, although if you're clever, you can leverage the knowledge you have (for instance, of outcomes of attacks) to spot new exploits of old problems.

4.3.1.1 Production/Expert Systems

One of the earliest schemes for performing misuse detection was the use of production/expert systems. This approach was utilized in systems such as MIDAS, IDES, Next Generation IDES (NIDES), DIDS, and CMDS. In the case of MIDAS, IDES, and NIDES, the production system employed was P-BEST, designed by Alan Whitehurst. For DIDS and CMDS, the CLIPS system, a public domain system developed by the National Aeronautics and Space Administration (NASA), was used.

The advantage associated with using production systems is that the control reasoning of the systems is separated from the statement of the problem solution. This feature allows users to enter knowledge about attacks as if-then rules and then enter facts (in the form of audit events); the system evaluates those facts according to the knowledge entered. This process can occur without the user ever affecting (or understanding) the internal function of the production system; before production systems, users had to hard code the decision engine and rules in custom software, a difficult, time-consuming task.

The attack knowledge is entered using an if-then syntax. Conditions indicative of an intrusion are specified on the left side (the "if" part) of the rule. When they are satisfied, the rule performs the actions on the right side (the "then" part) of the rule.

Some practical problems are associated with the use of production systems in intrusion detection.

- They are ill-equipped to handle large volumes of data. This is true because the declarative representations used by production systems are generally implemented as interpreted systems and interpreters are slower than compiled engines.
- They do not provide any natural handling of sequentially ordered data.
- The expertise reflected by the production system is only as good as the person on whose skills the system is modeled (the standard "garbage in, garbage out" effect endemic to computer systems).
- They can detect only known intrusions.
- They cannot handle uncertainty.

Maintaining the rule base can be problematic, as one must make changes to the rules considering the impact of the changes on the rest of the rules in the knowledge base.[3]

4.3.1.2 State Transition Approaches

State transition approaches to performing misuse detection structure the problem of misuse detection in a way that allows the use of optimized pattern-matching techniques. Their speed and flexibility make them among the most powerful intrusion detection capabilities at this time.

State transition approaches use expressions of system state and state transitions to describe and detect known intrusions. Several approaches are available for implementing state transition approaches to intrusion detection. The three major approaches are language or Application Programming Interface (API), characterization of state transitions, Colored Petri Nets (CP-Nets), and state transition analysis. In this section, I explore these processes, outlining the strategies each uses to characterize misuse patterns and then filter event data against them.

State Transition Analysis

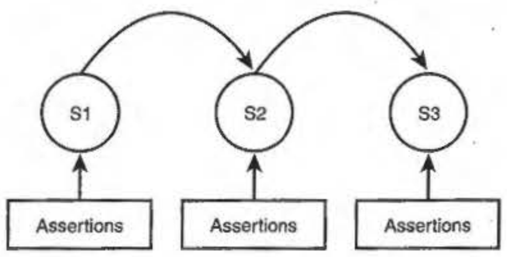
State transition analysis is an approach to misuse detection using high-level state transition diagrams to represent and detect known penetration scenarios. This approach was first explored in the STAT system and its extension to UNIX network environments, USTAT.

Both systems were developed at the University of California, Santa Barbara. Phillip Porras and Richard Kemmerer developed STAT; Koral Ilgun and Kemmerer developed USTAT.

A state transition diagram is a graphical representation of a penetration scenario. Figure 4.4 shows the components of a state transition diagram, as well as how they are used to represent a sequence. Nodes represent the states, and arcs represent the transitions. The concept of expressing intrusions in state transition form is rooted in the understanding that all intruders start with limited privileges and exploit system vulnerabilities to gain some outcome. Both the limited privilege starting point and the successful intrusion outcome can be expressed as system states.

In using state transition diagrams to characterize the intrusion sequences, the system can limit itself to expressing those key activities that result in a state change. The path between initial and intrusion state can be rather subjective; hence two persons can come up with different state transition diagrams that represent the same attack scenario. Each state consists of one or more state assertions (also shown in Figure 4.4).

Figure 4.4 State Transition Diagrams



State transition analysis systems utilize finite state machine graphs (finite automata) to model intrusions. The intrusion is composed of a sequence of actions that lead from some initial system state to an intrusion state. The initial state represents the state of the system before the intrusion is executed, and the intrusion state represents the state of the system upon the completion of the intrusion. The system state is described in terms of system attributes and/or user privileges. The transition is driven by a user action. The state transition engine maintains a set of state transition diagrams, each representing a penetration scenario. At a given time, it's assumed that some sequence of actions have driven the system to a particular state in each diagram. When a new action takes place, the engine checks it against each state transition diagram to see whether the action drives the scenario to the next state. If the action nullifies the assertions of the current state, the inference engine moves the state transition back to the nearest state for which the assertions still hold. If the action drives the scenario to the end state, indicating an intrusion, the previous transition information is sent to the decision engine, which alerts the security officer to the presence of the intrusion.

The advantages of the STAT approach follow:

- State transition diagrams provide an intuitive, high-level, and audit-record-independent representation of penetration scenarios.
- The transitions allow one to represent partial order of signature actions constituting an attack scenario.
- A state transition diagram uses the smallest possible subset of signature actions that must occur for the penetration to be successful. Thus, the detector can generalize over variants of the same penetration.
- The hard-link information maintained by the system makes it easier to express penetration scenarios.
- The system can detect coordinated and slow attacks.[4]

Deficiencies of the STAT approach include the following:

- The list of state assertions and signature actions are hand coded.
- The state assertions and signatures may not be powerful enough for expressing more elaborate penetration scenarios.
- The evaluation of certain state assertions may require the inference engine to get additional information from the target system. This process could cause performance degradation.
- The system cannot detect many common attacks, so it must be combined with other detectors for operational use.
- The prototyped system is slow compared to other state transition–based approaches.[5]

Colored Petri-Net and IDIOT

Another state-transition-based approach to optimizing misuse detection is the Colored Petri (CP)-Net approach developed by Sandeep Kumar and Gene Spafford at Purdue University. This approach was implemented in the IDIOT system.

Note

The name IDIOT originated as a joke between Gene Spafford and the author; it stands for Intrusion Detection In Our Time, a wry commentary on the delays many members of the intrusion detection research community experienced in transferring research results to operational systems!

IDIOT uses a variation of CP-Nets to represent and detect intrusion patterns. Under this model, an intrusion is represented as a CP-Net in which the color of tokens in each state serves to model the context of the event. The signature matching is driven by the audit trail and takes place by moving tokens progressively from initial states to the final state (indicating an intrusion or attack). *Guards* define the context in which signatures are considered matched, and *post actions* are performed when the pattern is successfully matched.

At first glance, this approach might appear to be almost identical to the state transition approach of STAT. However, there are significant differences between the approaches. First, in STAT the intrusion is detected by the effect it has on the system state, that is, the *outcome* of the intrusion. In IDIOT the intrusion is detected by pattern-matching the signature that constitutes the penetration. In STAT guards are placed in the state, whereas in IDIOT the guards are incorporated in the transitions.

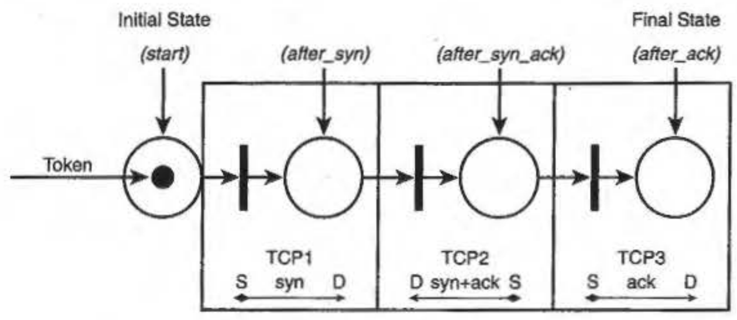
In IDIOT each intrusion signature is expressed as a pattern that represents the relationship among events and their context. This relationship pattern precisely represents a successful intrusion or its attempt. Vertices in the CP-Net graph represent system states. Intrusion patterns have preceding conditions and following actions associated with them. The scheme is independent of any underlying computational framework of matching and provides a model in which all categories in the classification are represented and matched.

This pattern-matching model consists of the following:

- A context representation that allows the matching to correlate various events that constitute the intrusion signature
- Semantics that accommodate the possibility of several intrusion patterns (possibly belonging to multiple event sources) being intermixed in the same event stream
- An action specification that provides for the execution of certain actions when the pattern is matched[2]

Figure 4.5 shows the CP-Net pattern for a TCP/IP connection (for a network connection not involving retransmissions).

Figure 4.5 CP-Net Pattern for TCP/IP Connection



The following are significant and numerous advantages associated with this approach to misuse detection:

- It is extremely fast. In experiments involving a nonoptimized version of IDIOT, for every hour of intense activity (generating C2 audit records), the detector required about 135 seconds to match about 100 intrusion patterns. This result represents a processing load of less than 5% for a Sun SPARCstation 5 generating about 6MB of audit data per hour.
- The pattern-matching engine is independent of audit format, so you can apply it to IP packet streams and other detection problems.
- The signatures are portable across audit trails so that they can be moved among different systems without having to rewrite them to accommodate vendor audit trail differences.
- The patterns can be specified according to what needs to be matched, not how it is to be matched.
- The sequencing and other ordering constraints on events can be represented directly.
- The system provides a fine-grained specification of a successful pattern match (that is, it tells you why an intrusion was detected).
- IDIOT provides a front-end language for encoding the graphical representation of the net.
- The system allows you to specify an action that is executed upon the matching of the signature, thereby supporting automated responses.

The limitations of IDIOT are those of all misuse detection systems. Although the capability to characterize intrusions in terms of outcomes allows you to generalize some detection signatures, the system still cannot detect new attacks that it doesn't know about.[6]

Language/API-Based Approaches

A common strategy for optimizing commercial misuse detection tools is to devise a means for describing intrusions in a form that a detection engine can use. Although, as we discussed before, some production expert system languages (such as P-BEST and CLIPS) are available, they were designed for other uses. Three approaches to expressing intrusions for misuse detection purposes are the RUSSEL language developed by Mounji at Faculties Universitaires Notre-Dame de la Paix (in Namur, Belgium), the STALKER system, patented by Smaha and Snapp of Haystack Laboratories, and the N packet filtering language developed by Marcus Ranum and provided as part of the Network Flight Recorder.

RUSSEL

RUSSEL is a rule-based language that is designed to optimize the processing of unstructured data streams, specifically operating system audit trails. It is optimized for heterogeneous system environments. The goal of RUSSEL is to enable users to correlate events across multiple hosts and to support multiple levels of abstraction for events. RUSSEL is utilized in ASAX, a misuse detection system optimized for heterogeneous network systems. ASAX features the use of a common audit data format (called the Normalized Audit Data Format [NADF]) and provides a component that supports adaptive rules.

Event patterns are expressed in RUSSEL as guarded actions of the form **Condition — > Action**. This action can be specified at a level of abstraction that allows the intrusion detection user to specify responses to a given detection scenario.

The language is structured as bottom up. It starts by asserting audit records as basic facts and then, based on these facts, tries to find audit record patterns that can be viewed as derived facts. The bottom-up structure is utilized because the audit records are not known at the time analysis starts and because it is more efficient than top-down strategies when dealing with large audit trails.[7]

STALKER

Another approach to characterizing intrusion for misuse detection purposes is the approach utilized in the STALKER system, a commercial misuse detection product.[8] This patented approach utilizes a common audit data format (the SVR 4++ standard outlined in Chapter 7, "Technical Issues") and a state-based data structure for attack signatures.

The detector is implemented as a finite state machine (the underlying technology for compilers), which is optimized to pattern-matching tasks such as misuse detection.

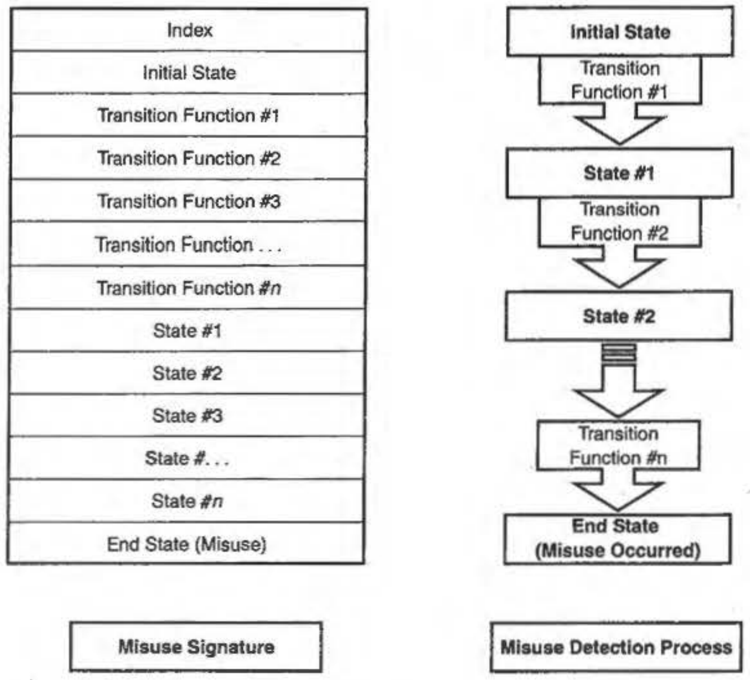
The misuse detector operates by passing audit records to the misuse detection engine. The engine maintains a set of detection signatures in the form of state transitions. The signature expression consists of a data structure that contains an initial state, an end state, and one or more sets of transition functions for each misuse. Figure 4.6 shows the structure and operation of the STALKER misuse detector.

This approach was successfully implemented and fielded in a series of commercial products, which supported a variety of operating systems and applications environments. The STALKER product was withdrawn from the market when Network Associates acquired Haystack Laboratories in 1997, but the design has recently been returned to the market in the Cybercop Monitor.[9]

Network Flight Recorder—N-Code

A language-based optimization of network monitoring and analysis functions is provided by the Network Flight Recorder, a network monitoring system that serves as the basis for some commercial network-based intrusion detection products. The N programming language is an interpreted language operating on a byte-code instruction and implementing a simple stack machine.

Figure 4.6 STALKER Misuse Detection Approach



The language includes flow control, procedures, and 64-bit integer counter data types. It is customized to support network packet filter construction. Although these packet filters can be programmed to recognize network attacks, they can also be programmed to recognize other network activity.

Filters are written in N-code, which is input to the network monitoring engine, compiled, and stored as byte-code instructions to optimize filtering performance. Network packet traffic is reassembled using a table structure, which preserves the state of each current network session. This state preservation permits matching signatures against the entire lifetime of a connection or traffic stream.

The underlying engine also keeps statistics about the network performance, including timing statistics regarding packet arrival rate and network errors (as evidenced by broken packets or duplicate packet traffic). The engine also uses a statistical calculation to determine how long to retain state information about a connection before discarding it. In hybrid intrusion detection systems, this statistical information can be passed to an anomaly detection component.

Under the N-code packet filtering mechanism, the language binds a filter to the reception of a network packet. It can also support other specified events. The filter collects information from the packet, exporting it through either an alert or record mechanism. Alert mechanisms send alert messages to an alert management system, whereas record mechanisms send a data structure to a recorder function for various types of additional back-end processing.

The N language is included in the Network Flight Recorder product, which is available in source code form from the developers.[10]

4.3.1.3 Information Retrieval for Batch Mode Analysis

Although most current intrusion detection is based on real-time collection and analysis of event data, some approaches involve working with audit data archives, searching for evidence of interesting patterns of activity, or providing the ability to isolate the activities affecting a particular object or involving a particular user. This is of special interest to investigators and incident handlers.

One such approach comes from Ross Anderson and Abida Khattak of the University of Cambridge. They propose a functional separation between intrusion detection systems that discover new attacks and systems that allow security administrators to find instances of the attacks after they are identified. To handle the latter problem, Anderson and Khattak propose the use of information retrieval (IR) techniques. These techniques are currently widely utilized in the form of search engines (such as AltaVista) for the World Wide Web.

IR systems utilize inverted files as indexes that allow efficient searching for keywords and combinations of keywords. These systems also utilize algorithms that learn a user's preferences and use Bayesian inference to help refine searches. These differ from data mining in that they rely on indexing, not machine learning, to discover patterns of data.

This approach is limited to reviewing audit information after the fact (in batch mode). The researchers constructed a prototype utilizing the UNIX `lastcomm` system log and GLIMPSE, a search engine developed by the University of Arizona. They used a Perl script to sort the file into a selection of smaller files, one for each user. A GLIMPSE search was entered, searching for command-line sequences that matched the sequences associated with a particular attack. This process quickly and reliably located the attacks.

This approach is simple, yet powerful, and significant efficiencies are associated with the use of the IR techniques to perform the detection activity. The index utilized by GLIMPSE is compact (about 2%–4% of the indexed material) and can serve as an effective data reduction mechanism for audit trails. The presence of the index can also serve as a fail-safe mechanism, revealing situations in which hackers alter audit information to cover their tracks. Because GLIMPSE and proposed audit data sources are either free or included with standard operating system packages, this approach is also inexpensive, an advantage for many organizations that cannot afford other solutions.[11]

4.3.2 Anomaly Detection

Anomaly detection involves a process of establishing profiles of normal user behaviors, comparing actual user behavior to those profiles, and flagging deviations from the normal. The basis of anomaly detection is the assertion that abnormal behavior patterns indicate misuse of systems. *Profiles* are defined as sets of metrics. *Metrics* are measures of particular aspects of user behavior. Each metric is associated with a threshold or range of values.

Anomaly detection depends on an assumption that users exhibit predictable, consistent patterns of system usage. The approach also accommodates adaptations to changes in user behavior over time. The completeness of anomaly detection has yet to be verified (no one knows whether any given set of metrics is rich enough to express all anomalous behavior). Thus, additional research is required to know whether anomaly detection will ever be able to detect all scenarios of interest, representing a strong protection mechanism for systems.

4.3.2.1 Denning's Original Model

Dorothy Denning, in her landmark 1986 paper outlining the IDES model for intrusion detectors, asserts that four statistical models may be included in the system. Each model is deemed suitable for a particular type of system metric.

Operational Model

First is the *operational model*. This model applies to metrics such as event counters for the number of password failures in a particular time interval. The model compares the metric to a set threshold, triggering an anomaly when the metric exceeds the threshold value. This model, which applies to misuse detection as well as anomaly detection, corresponds to threshold detection, covered later in this section.

Mean and Standard Deviation Model

Denning's second detection model proposes a classic mean and standard deviation characterization of data. The assumption is that all the analyzer knows about system behavior metrics are the mean and standard deviations as determined from the first two moments. A new behavior observation is defined to be abnormal if it falls outside a confidence interval. This confidence interval is defined as *d* standard deviations from the mean for some parameter *d*. Denning hypothesizes that this characterization is applicable to event counters, interval timers, and resource measures. She also alludes to the ability to assign weights to these computations, such that more recent data is assigned a greater weight.

Multivariate Model

The multivariate model, the third of Denning's detection models, is an extension to the mean and standard deviation model. It is based on performing correlations among two or more metrics. Therefore, instead of basing the detection of an anomaly strictly on one measure, you might base it on the correlation of that measure with another measure. So instead of detecting an anomaly based solely on the observed length of a session, you might base it on the correlation of the length of the session with the number of CPU cycles utilized.

Markov Process Model

The final, most complex part of Denning's model is limited to event counters. Under this model, the detector considers each different type of audit event as a state variable and uses a state transition matrix to characterize the transition frequencies between states (not the frequencies of the individual states/audit records). A new observation is defined as anomalous if its probability, as determined by the previous state and value in the state transition matrix, is too low. This allows the detector to spot unusual command or event sequences, not just single events. This introduces the notion of performing stateful analysis of event streams.[12]

4.3.2.2 Quantitative Analysis

The most commonly used anomaly detection approach is quantitative analysis in which detection rules and attributes are expressed in numeric form. Denning includes this category of measures in her operational model. This set of techniques often presumes some computation, which can range from simple addition to more complex cryptographic calculations. The results of these techniques can be the basis for misuse detection signatures and anomaly detection statistical models alike. This section describes several common quantitative analyses and provides an example of an operational system that utilizes these techniques to accomplish data reduction and intrusion detection goals.

Threshold Detection

Probably the most common form of quantitative analysis is *threshold detection* (also known in some circles as *thresholds and triggers*). In threshold detection, certain attributes of user and system behaviors are characterized in terms of counts, with some level established as permissible. The classic example of a threshold is the number of permissible unsuccessful logins to a system. Virtually every early intrusion detection system contained a detection rule defining an intrusion in terms of this measure.

Other thresholds include the number of network connections of a particular type, the number of attempted file accesses, the number of files or directories accessed, and the number of network systems accessed. An inherent assumption in threshold detection is that the measurement is made over a particular time interval. This interval can be fixed in time (for instance, the threshold can be reset to zero at a particular time of day) or function over a sliding window (for example, the measurement is made over the last eight hours).

Heuristic Threshold Detection

Heuristic threshold checks take simple threshold detection a step further by adapting it to observed levels. This process increases the accuracy of the detection, especially when performing detection over a wide range of users or target environments. So, for instance, instead of having a threshold detection rule triggering an alert when the number of failed logins exceeds three in an eight-hour period, you can have a threshold detection rule that triggers an alert when an abnormal number of failed logins occur. "Abnormal" can be defined by various formulas. One that comes immediately to mind is a Gaussian function (such as chi-square) in which the mean number of failed logins is calculated and subsequent numbers of failed logins are compared to the mean plus some standard deviation.

Target-Based Integrity Checks

Another valuable quantitative analysis measure is a target-based integrity check. This is a check for a change in a system object that should not be subject to unpredictable change. The most common example of such an integrity check utilizes a message digest function to calculate a cryptographic checksum of the system object in question. After the checksum is calculated and stored in a safe place (for instance, read-only media) the system periodically recalculates the checksum, comparing it to the stored reference value. If a differential is found, an alarm is raised. The Tripwire™ product, found in both public domain and commercially supported versions, provides this capability.

Quantitative Analysis and Data Reduction

One of the more interesting uses of quantitative analysis in early intrusion detection systems used quantitative measures to perform data reduction. *Data reduction* is the process of eliminating superfluous or redundant information from often-voluminous event information. This reduces system storage loads and optimizes detection processes based on the event information.

An example demonstrating the use of quantitative methods to support effective data reduction comes from the NADIR system, developed by the Computing and Communications Division of Los Alamos National Laboratory. The NADIR developers utilized *data profiling*, which transforms user activity from audit logs into vectors of quantitative measures (most of them linear categorical or a combination of linear categorical and ordinal data). The profiles are aggregated over time (with weekly summaries) as well as over systems (with aggregate views of user activity per system). The reduced data is subjected to both statistical and expert system examination, with alarms and alerts handled by a staff investigator.[13]

4.3.2.3 Statistical Measures

The first worked examples of anomaly detection systems were based on statistical measures. These included approaches such as those utilized in IDES, mentioned earlier, and the follow-on NIDES project, as well as the Haystack system.

IDES/NIDES

IDES and NIDES, developed by researchers at SRI International, were two of the most prominent early intrusion detection research systems. They were both hybrid systems, including misuse and anomaly detection features; however, I focus on the statistical analysis here.

The statistical analysis techniques employed for IDES and NIDES support historical statistical profiles established and maintained for each user and system subject. These profiles are updated periodically, with older data aged so that the profiles adapt to reflect changes in user behavior over time.

The systems maintain a statistical knowledge base consisting of profiles. Each profile expresses normal behaviors for a particular user in terms of a set of measures or metrics. Once a day, new audit data is incorporated into the knowledge base (after the old vectors are aged by an exponential decay factor), based on the activity of the user during that day.

Let's take a look at how these statistics were calculated for IDES. Each time an audit record is generated, a summary test statistic is generated. This statistic, called the IDES score (IS), is calculated by the following formula:

$$IS = (S_1, S^2, S_3 \ldots S_n) C^{-1} (S_1, S_2, S_3 \ldots S_n) t;$$

where `(S..) C-1` is the inverse of a correlation matrix or vector and `(S...) t` is the transpose of the vector. Each S_n measures some aspect of behavior, such as file access, terminals used, and CPU time used. Different S_n values can also represent different views of the same aspect of behavior.[14]

Haystack

Haystack, an anomaly detection system developed by Tracor Applied Sciences and Haystack Laboratories for the U.S. Air Force, employs a two-part statistical anomaly detection approach. The first measure determines to what degree a user session resembles an established intrusion type. This measure is calculated as follows:

1. The system maintains a vector of user behavior measures.
2. For each type of intrusion, the system associates a weight with each behavior measure, reflecting the relevance of the measure to the given intrusion type.
3. For each session, the vector of user behavior measures is calculated and compared to the threshold vector.
4. Those behavior measures for which threshold settings are exceeded are noted.
5. The weights associated with the measures exceeding threshold are summed.
6. The sum is used to assign a suspicion quotient to the session, based on the distribution of the weighted intrusion scores for all previous sessions.

The second, complementary statistical method detects deviations in a user's session activities from the normal user session profile. This method looks for session statistics that significantly deviate from the normal historical statistical profile for that user.[15, 16]

Strengths of Statistical Analysis

Statistical anomaly detection analysis originally targeted intruders masquerading as legitimate users. Although the assertion has been made that statistical analysis may also detect intruders who exploit previously unknown vulnerabilities who could not be detected by any other means, this assertion has yet to be proven in production use of a system. Early researchers also hypothesized that statistical anomaly detection could reveal interesting, sometimes suspicious, activities that could lead to discoveries of security breaches. This assertion was confirmed on at least one system, NADIR (in operation at Los Alamos National Laboratory), where developers reported that some of the information gained by using NADIR led to the discovery of system and security process errors, as well as discoveries that allowed them to improve the general management of Los Alamos's system complex.[17]

Another advantage often claimed for statistical analysis is that statistical systems do not require the constant updates and maintenance that misuse detection systems do. This claim may be true, but it depends on several factors. Metrics must be well chosen, adequate for good discrimination, and well-adapted to changes in behavior (that is, changes in user behavior must produce a consistent, noticeable change in the corresponding metrics). If these conditions are met, chances are excellent that the statistical analyzer will reliably detect behaviors of interest without requiring on-the-fly modifications to the system.

Drawbacks of Statistical Analysis

On the other hand, statistical analysis systems have significant drawbacks. First, most were designed to perform batch mode processing of audit records, which eliminated the capability to perform automated responses to block damage. This omission was not a problem at the time the systems were proposed because early systems were designed to monitor audit trails from centralized, mainframe target platforms. Although later systems attempted to perform real-time analysis of audit data, the memory and processing loads involved in using and maintaining the user profile knowledge base usually caused the system to lag behind audit record generation.

A second drawback affects the range of events that statistical analysis can characterize. The nature of statistical analysis precludes the capability to take into account the sequential relationships between events. The exact order of the occurrence of events is not provided as an attribute in most of these systems. In other words, the event horizon for these systems is limited to one event. Because many anomalies indicating attack depend on such sequential event relationships, this situation represents a serious limitation to the approach.

In cases when quantitative methods (Denning's operational model) are utilized, it is also difficult to select appropriate values for thresholds and ranges.

The false alarm rates associated with statistical analysis systems are high, which leads to users ignoring or disabling the systems. These false alarms include both type 1 (false negative) and type 2 (false positive) errors.

4.3.2.4 Nonparametric Statistical Measures

Early statistical approaches were similar in that they utilized *parametric* approaches to characterizing the behavioral patterns of users and other system entities. *Parametric approaches* refer to analytical approaches in which assumptions are made about the underlying distribution of the data being analyzed. For instance, in early versions of IDES and MIDAS the distributions of user usage patterns were assumed to be Gaussian or normal.

The problem with these assumptions is that error rates are high when the assumptions are incorrect. When researchers began collecting information about system usage patterns that included attributes such as system resource usage, the distributions were discovered not to be normal, and including these measures led to high error rates.

Linda Lankewicz and Mark Benard of Tulane University proposed that a way of overcoming these problems was to utilize nonparametric techniques for performing anomaly detection. This approach provides the capability to accommodate users with less predictable usage patterns and allows the analyzer to take into account system measures that are not easily accommodated by parametric schemes.

The approach Lankewicz and Benard utilized involved nonparametric data classification techniques, specifically *clustering analysis.* In clustering analysis, large quantities of historical data are collected (a *sample set*) and organized into *clusters* according to some evaluation criteria (also known as *features*). Preprocessing is performed in which features associated with a particular event stream (often mapped to a specific user) are converted into a vector representation (for example, $Xi = [f_1, f_2, \ldots f_n]$ in an n-dimensional state). A clustering algorithm is used to group vectors into classes of behaviors, attempting to group them so that members of each class are as close as possible to each other while different classes are as far apart as they can be.

In nonparametric statistical anomaly detection, the premise is that a user's activity data, as expressed in terms of the features, falls into two distinct clusters: one indicating anomalous activity and the other indicating normal activity.

Various clustering algorithms are available. These range from algorithms that use simple distance measures to determine whether an object falls into a cluster, to more complex concept-based measures (in which an object is "scored" according to a set of conditions and that score is used to determine membership in a particular cluster). Different clustering algorithms usually best serve different data sets and analysis goals.

Researchers at Tulane found that the clustering algorithm that best accomplished this goal using resource usage figures as evaluation criteria was the *k-nearest-neighbor* algorithm. This groups each vector with *k* of its nearest neighbors. *k* is a function of the number of vectors in the sample set, not a fixed value.

Experimental results using this analysis technique showed that clusters formed that reliably grouped similar system operations (such as compiling or editing files) and also grouped activity patterns according to user.

The advantages of nonparametric approaches include the capability to perform reliable reduction of event data (in the transformation of raw event data to vectors). This reduction effect was documented as more than two orders of magnitude. Other benefits are improvement in the speed of detection and improvement in accuracy over parametric statistical analysis. Disadvantages involve concerns that expanding features beyond resource usage would lessen the efficiency and accuracy of the analysis.[18]

4.3.2.5 Rule-Based Approaches

Another variation of anomaly detection is rule-based anomaly detection. The assumptions underlying this approach are the same as those associated with statistical anomaly detection. The main difference is that rule-based intrusion detection systems use sets of rules to represent and store usage patterns. Two such approaches are covered in this section: the Wisdom and Sense approach and the Time-Based Inductive Machine (TIM).

Wisdom and Sense

The first rule-based anomaly detection system was the Wisdom and Sense (W&S) system, developed by researchers at Los Alamos National Laboratory and Oak Ridge National Laboratory. W&S can operate on a variety of system platforms and can characterize activity at both operating system and application levels. It provides two schemes for populating rule bases: entering them manually (to reflect a policy statement) and generating them from historical audit data. The rules are derived from historical audit data by performing a categorical examination, expressing the patterns found in terms of rules. The rules reflect the past behavior of system subjects and objects and are stored in a tree structure, called a *forest*. Specific data values within the audit records are grouped into *thread classes* with which collections of operations or rules are associated.

An example of a thread class is "all the records containing the same user-file field values." Rules are applied to the data in a thread each time an activity associated with that thread occurs. Anomalies are detected this way: When transactions are processed, they are compared to the events of the matching thread to determine whether the events match the historical patterns of activity or represent an anomaly.[19]

TIM

The TIM system, proposed by Teng, Chen, and Lu, while they were associated with the Digital Equipment Corporation, utilizes an inductive approach to dynamically generate rules defining intrusion. The difference between TIM and other anomaly detection systems is that TIM looks for patterns in *sequences* of events, not in individual events. It effectively implements a Markov transition probability model, as proposed by Denning in her seminal intrusion detection work.

TIM observes historical event record sequence, characterizing the probability of particular sequences of events occurring. Other anomaly detector systems measure whether or not the occurrence of a single event represents a deviation from normal patterns of activity. TIM focuses on sequences of events, checking to see whether a chain of events corresponds to what would be expected based on its observation of historical event sequences.

For example, suppose events E_1, E_2, and E_3 are listed sequentially in an audit trail. TIM characterizes the probability of the occurrence of E_1 followed by E_2 followed by E_3, based on the history of sequences it has observed in the past. TIM automatically generates rules about the event sequences as it analyzes historical event data, and then stores the rules in a rule base. Because TIM groups event sequences, the amount of space required for the rule base is significantly smaller than that required for a single-event-oriented rule based system (such as Wisdom and Sense).

If a sequence of events matches the head of a rule, then the next event is considered anomalous if it's not in the set of predicted events in the body of the rule. The system also refines its analysis by deleting less predictive rules from the rule base. (Rule 1 is *more predictive* than rule 2 if rule 1 successfully predicts more events than rule 2 predicts.)

The advantages for TIM, especially when compared to statistical measures, are significant. This approach is well suited to environments where user patterns differ significantly from user to user, but where each user exhibits consistent behavior over time. Such an environment might be represented by a large corporation in which different users are responsible for accounting, administrative, programming, and personnel functions with very little crossover of user duties. This approach is also well suited for environments in which threat is associated with a few event types rather than the full complement of system events. Finally, this approach is not subject to problems associated with *session creep*, a defeat strategy associated with anomaly detection, in which an attacker gradually alters his/her behavior pattern over time to train the system to accept intrusive behavior as normal. This resistance to session creep attacks is due to the fact that the semantics are built into the detection rules.

However, as with other systems, weaknesses are associated with the TIM approach. It suffers the problem associated with all learning-based approaches in that the effectiveness of the approach depends on the quality of the training data. In learning-based systems the training data must reflect normal activity for the users of the system. Furthermore, the rules generated by this approach may not be comprehensive enough to reflect all possible normal user behavior patterns. This weakness produces a large false positive (type 2) error rate, especially at the beginning of the operation of the system. The error rate is high because if an event does not match the head of any rule (that is, if the system did not encounter the event type in the training data set), that event always triggers an anomaly.

This approach served as the basis for the Digital Equipment Corporation Polycenter intrusion detection product and forms the foundation for much subsequent anomaly detection research.[20]

4.3.2.6 Neural Networks

Neural networks use adaptive learning techniques to characterize anomalous behavior. This nonparametric analysis technique operates on historical sets of training data, which are presumably cleansed of any data indicating intrusions or other undesirable user behavior.

Neural networks consist of numerous simple processing elements called *units* that interact by using weighted *connections*. The knowledge of a neural network is encoded in the structure of the net in terms of connections between units and their weights. The actual learning process takes place by changing weights and adding or removing connections.

Neural network processing involves two stages. In the first stage (corresponding to the "building the detector" stage of the intrusion analysis model outlined earlier in this chapter), the network is populated by a training set of historical or other sample data that is representative of user behavior. In the second stage (corresponding to the second stage of the intrusion analysis model), the network accepts event data and compares it to historical behavior references, determining similarities and differences.

The network indicates that an event is abnormal by changing the state of the units, changing the weights of connections, adding connections, or removing them. The network also modifies its definition of what constitutes a normal event by performing stepwise corrections.

Neural network approaches hold a great deal of promise for anomaly detection. Because they don't use a fixed set of features to define user behaviors, feature selection is irrelevant. Neural networks don't make prior assumptions on expected statistical distribution of metrics, so this method retains some of the advantages over classic statistical analysis associated with other nonparametric techniques.

Among the problems associated with utilizing neural networks for intrusion detection is a tendency to form mysterious unstable configurations in which the network fails to learn certain things for no apparent reason. However, the major drawback to utilizing neural networks for intrusion detection is that neural networks don't provide any explanation for the anomalies they find. This practice impedes the ability of users to establish accountability or otherwise address the roots of the security problems that allowed the detected intrusion. This made it poorly suited to the needs of security managers. Although some researchers have proposed hybrid approaches as a means of overcoming these disadvantages, no published figures yet indicate the feasibility of neural network approaches.[21]

4.3.3 Alternative Detection Schemes

Some recent intrusion detection approaches fit neither misuse detection nor anomaly detection categories. These schemes may be applicable to either problem, perform precursor activity that can drive or refine either form of detection, or depart from the traditional monolithic view of intrusion detection in ways that affect detection strategies.

4.3.3.1 Immune System Approaches

In an innovative and promising research project, researchers at the University of New Mexico (Forrest, Hofmeyr, and Somayagi, among others) took a fresh look at the entire question of computer security. The question posed by the researchers was, "How does one equip computer systems with the means to protect themselves?" In answering this question, they noted marked similarities between biological immune systems and system protection mechanisms.

The key to both systems' functioning well is the capability to perform "self/nonself" determination—that is, the capability of an organism's immune system to determine which materials are harmless entities (such as the organism's own) and which are pathogens and other dangerous factors. As the immune system performs this determination by using peptides, short protein fragments, the researchers decided to focus on some computer attribute that could be considered analogous to peptides. The team hypothesized that sequences of UNIX system calls could satisfy those requirements.

In deciding to consider system calls as a primary source of information, the researchers considered a variety of goals for the data, including data volume, capability to reliably detect misuse, and suitability for encoding in a fashion appropriate for advanced pattern-matching techniques. They chose to focus on short sequences of the system calls, furthermore ignoring the parameters passed to the calls, looking only at their temporal orders.

The system as first proposed performs anomaly detection. (It can also perform misuse detection.) The system complies with the two-phase intrusion detection analysis process, with the first phase building a knowledge base that profiles normal behavior. This profile is a bit different from others discussed in this chapter in that here the behavior characterized is not user-centric, but system-process centric. Deviations from this profile are defined as anomalous. In the second phase of the detection system, the profiles are used to monitor subsequent system behavior for anomalies.

Sequences of system calls that result from running privileged processes were collected over time. The profiles for the system consisted of unique sequences of length 10. Three measures were utilized to characterize deviations from normal process behaviors: successful exploits, unsuccessful exploits, and error conditions.

The results were extremely promising because the three measures allowed the detection of several sorts of anomalous behavior spanning several historically problematic UNIX programs. The research also showed that the sets of execution sequences were remarkably compact.[22]

Subsequent research compared different approaches for characterizing normal behavior. It explored whether more powerful data modeling methods significantly improved the performance of this approach when monitoring more complex systems. Somewhat surprisingly, even powerful data modeling techniques (for example, hidden Markov models, which are extremely reliable, though computationally greedy) did not give significantly better results than the simpler time-sequence-based models.

Although the self/nonself techniques appear to constitute an extremely powerful and promising approach, it is not a complete solution to the intrusion detection problem. Some attacks, including race conditions, masquerading, and policy violations, do not involve the use of privileged processes. Therefore, these attacks are not subject to detection using this approach.[23]

4.3.3.2 Genetic Algorithms

Another more sophisticated approach to performing anomaly detection utilizes *genetic algorithms* to perform analysis of event data.

A genetic algorithm is an instance of a class of algorithms called *evolutionary algorithms.* Evolutionary algorithms incorporate concepts of Darwinian natural selection (survival of the fittest) to optimize solutions to problems. Genetic algorithms utilize encoded forms (known as *chromosomes*) with methods that allow combination or mutation of the chromosomes to form new individuals. These algorithms are recognized for their capability to deal with multidimensional optimization problems in which the chromosome is composed of encoded values for the variables being optimized.[24]

In the eyes of the researchers investigating genetic algorithm approaches to intrusion detection, the intrusion detection process involves defining hypothesis vectors for event data, where the vector either indicates an intrusion or does not. The hypothesis is then tested to determine whether it is valid, and an improved hypothesis is devised and tried based on the results of the test. This process repeats until a solution is found.

The role of genetic algorithms in this process is to devise the improved hypothesis. Genetic algorithm analysis involves two steps. The first step involves coding a solution to the problem with a string of bits. The second step is finding a fitness function to test each individual of the population (for instance, all the possible solutions to the problem) against some evaluation criteria.

In the system GASSATA, developed by Ludovic Mé of Supeleć, the French engineering university, genetic algorithms are applied to the problem of classifying system events by using a set of hypothesis vectors, H (one vector per event stream of interest) of n dimensions (where n is the number of potential known attacks). H_i is defined to be 1 if it represents an attack and 0 if it doesn't.

The fitness function has two parts. First, the risk that a particular attack represents to the system is multiplied by the value of the hypothesis vector. The product is then adjusted by a quadratic penalty function to eliminate unrealistic hypotheses. This step improves the discrimination among the possible attacks. The goal of the process is to optimize the results of this analysis so that the probability of a detected attack being real approaches 1 and the probability of a detected attack being false approaches 0.

Experimental results for the genetic algorithm approach to anomaly detection are encouraging. In experimental runs, the mean probability of true positives (accurate detection of real attacks) was 0.996, and the mean probability of false positives (detection of nonattacks) was 0.0044. The time required to construct the filters is also encouraging. For a sample set of 200 attacks, it took the system 10 minutes and 25 seconds to evaluate audit records generated by an average user over 30 minutes of intensive system use.

The following drawbacks are noted in this approach to misuse detection:

- The system can't take into account attacks characterized by event absence (for instance, rules in the form of "programmer does NOT use cc as compiler").
- Because of the binary expressional form for individual event streams, the system can't detect multiple simultaneous attacks. The possibility exists that nonbinary genetic algorithm approaches could solve this problem.
- If the same event or group of events is common to several attacks and an attacker uses this commonality to execute multiple simultaneous attacks, the system can't find an optimal hypothesis vector.
- Perhaps the largest drawback is that the system doesn't precisely locate attacks in the audit trail. Therefore, no sense of temporality occurs in the results of the detector.

(Note that this drawback is similar to problems noted in neural network approaches.) Therefore, genetic algorithm approaches must be backed up with post hoc search or other investigative aids if such support is required.[25]

4.3.3.3 Agent-Based Detection

Agent-based approaches to intrusion detection are based on software entities that perform certain security monitoring functions at a host. They function autonomously—that is, they are controlled only by the operating system, not by other processes. Agent-based approaches also run continuously with the understanding that this type of operation allows them to learn from experiences, as well as to communicate and cooperate with other agents of similar construction.

Agent-based detection approaches can be very powerful due to the range of capabilities with which one can imbue an agent. An agent can be extremely simple (for example counting the number of times a particular command is invoked in a time interval) or complex (looking for evidence of a set of attacks for a particular environment), depending on the whim of the developer.

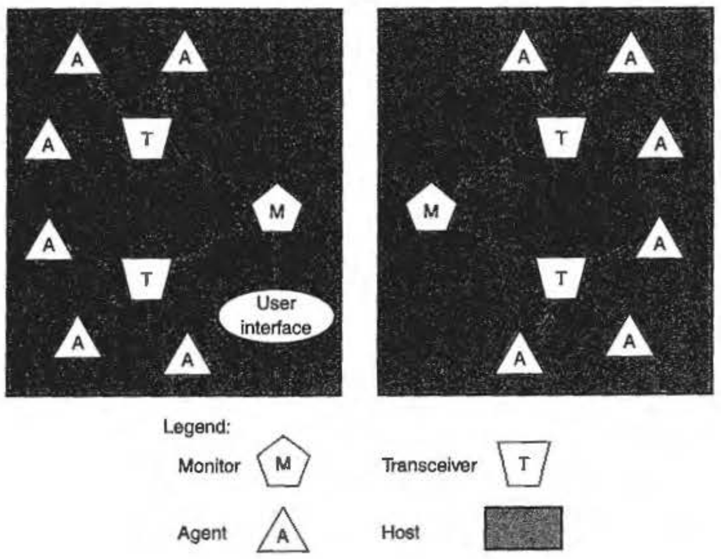
The range of agent capabilities can allow an agent-based intrusion detection system to provide a mixture of anomaly detection and misuse detection capabilities. For instance, an agent can be programmed to adapt its detection capabilities to changes in the local environment. It can also monitor for very subtle patterns over a long time interval, thereby detecting slow attacks. Finally, an agent can enact extremely fine-grained responses to a detected problem (for instance, changing the priority level of a process, effectively slowing it down).

Autonomous Agents for Intrusion Detection

A prototype of an agent-based intrusion detection system, Autonomous Agents for Intrusion Detection (AAFID), was developed by researchers at Purdue University. It serves as the basis for this discussion of agent-based solutions.

This architecture for agent-based intrusion detection systems calls for a hierarchically ordered control and reporting structure for agents, as pictured in Figure 4.7. Any number of agents can reside on a host. All agents on a particular host report their findings to a single transceiver. Transceivers monitor the operation of all the agents running in the host, with capabilities to issue start, stop, and reconfiguration commands to agents. Transceivers also perform data reduction on information reported by the agents and report results to one or more monitors, the next level of the hierarchy.

Figure 4.7 AAFID Architecture



Monitors, which can be hierarchically structured in multiple layers, control and consolidate information from several transceivers. The architecture of AAFID allows redundancy in reporting from transceivers to monitors so that the failure of a monitor doesn't jeopardize the operation of the intrusion detection system. Monitors have the capability to access data from the entire network and therefore perform higher-level aggregation of results from transceivers. This feature enables the system to detect multihost attacks. Through a user interface, the user of the system enters commands to control the monitors. They, in turn, control the transceivers based on these user commands.

APIs exported by each component accomplish communications between agents, transceivers, monitors, and users.

The advantages of AAFID and other such agent-based approaches include the following:

- They appear to be more resistant to insertion and evasion attacks than other intrusion detection systems.
- The architecture is more easily scaled, with provisions for adding new components or replacing old ones as needed.
- Agents can be tested independent of the full system before they are deployed.
- Because they can intercommunicate, agents can be deployed in groups in which each performs a different, simple function but contributes to a complex result.

The following deficiencies are also associated with the AAFID architecture:

- Monitors are single points of failure. If a monitor ceases to work, all the transceivers under its control stop producing useful information. Possible solution strategies exist, but they are as yet untested.
- If duplicated monitors are used to address the first problem, it is difficult to deal with the consistency and duplication of information. This situation requires additional mechanisms.
- Access control mechanisms to allow different users to have different modes of access to the intrusion detection system are missing. This significant deficit must be addressed at each level of the architecture.
- Problems occur because of the propagation time for evidence of attackers to reach the monitor level. This problem is common to all distributed intrusion detection systems.
- As in the rest of intrusion detection, a significant need remains for more insight regarding designing user interfaces for intrusion detection systems. This need extends from presentation schemes to policy structures and specification schemes.[26]

EMERALD

A second architecture that utilizes distributed agent approaches to performing intrusion detection is the EMERALD system, researched and prototyped by Phillip Porras (whose previous intrusion detection work includes the STAT state transition analysis system) of SRI International. EMERALD includes numerous local monitors in a framework that supports distributing local results to a global array of detectors that, in turn, consolidate alarms and alerts.

EMERALD, like IDES and NIDES, SRI's previous intrusion detection systems, is a hybrid intrusion detector, utilizing both signature analysis and statistical profiling to detect security problems.

EMERALD is notable in that it separates the analysis semantics from the analysis and response logic, thereby enabling much easier integration throughout the network. EMERALD also supports analysis at different layers of abstraction, an important capability. Futhermore the design supports interoperability, another important issue in modern intrusion detection systems.

The central component of EMERALD is the EMERALD Service Monitor, which is similar in form and function to the AAFID autonomous agent. Service monitors are programmable to perform any function and are deployed to hosts. They can be layered, to support hierarchical data reduction, with service monitors performing some local analysis and reporting results and additional information to higher-level monitors. This approach yields some scalability not commonly found in network intrusion detection products. The system also supports a wide range of automated response, which is of great interest to customers responsible for critical large-scale networks.

EMERALD, as it builds on considerable corporate insight gathered in the IDES and NIDES efforts, holds great promise for protecting large distributed networks.[27, 28]

4.3.3.2 Data Mining

An approach that is similar to some of the rule-based anomaly detection efforts involves utilizing data mining techniques to build intrusion detection models. The objective of this approach is to discover consistent useful patterns of system features that can be used to describe program and user behaviors. These sets of system features, in turn, can then be processed by inductive methods to form classifiers (detection engines) that can recognize anomalies and misuse scenarios.

Data mining refers to the process of extracting models from large bodies of data. These models often discover facts in the data that are not apparent through other means of inspection. Although many algorithms are available for data mining purposes, the three that are most useful for mining audit data are *classification, link analysis,* and *sequence analysis.*

- *Classification* assigns a data item to one of several predefined categories. (This step is akin to sorting data into "bins," depending on some criteria.) Classification algorithms output *classifiers,* such as decision trees or rules. In intrusion detection, an optimal classifier can reliably identify audit data as falling into a normal or abnormal category.
- *Link analysis* identifies relationships and correlations between fields in the body of data. In intrusion detection, an optimal link analysis algorithm identifies the set of system features best able to reliably reveal intrusions.
- *Sequence analysis* models sequential patterns. These algorithms can reveal which audit events typically occur together and hold the key to expanding intrusion detection models to include temporal statistical measures. These measures can provide the capability to recognize denial-of-service attacks.

Researchers have developed extensions to standard data mining algorithms to accommodate some of the special needs of audit and other system event logs. Initial results of experiments using live data are interesting, but the work is not yet ready for transfer into commercial products. Additional research is planned to refine the approach.[29]

4.4 Conclusion

In this chapter, you've seen the different approaches to the core function of intrusion detection: analysis. In analysis, which involves isolating patterns of behavior known to represent problems (misuse detection) and using mathematical approaches to characterize user behaviors that are abnormal (anomaly detection), intrusion detection systems address several system protection challenges.

Although many approaches have been explored for doing both misuse and anomaly detection, most commercial products confine themselves to performing pattern matching and elementary statistical characterization of user activity and usage patterns. Refining advanced analysis techniques and transferring them to commercial products represent significant challenges for the intrusion detection research and development community. However, this step is necessary if intrusion detection products are to be effective in protecting real systems, both now and in the future.

Endnotes

1. Lunt, T., A. Tamaru, and F. Gilham. "IDES: A Progress Report." Proceedings of the Sixth Annual Computer Security Applications Conference, Tucson, AZ, December 1990.
2. Ranum, M. personal communication, June 1999.
3. Kumar, S. "Classification and Detection of Computer Intrusions." Ph.D. thesis, Department of Computer Sciences, Purdue University, 1995.
4. Ilgun, K. "USTAT: A Real-Time Intrusion Detection System for UNIX." Master thesis, Computer Science Department, University of California, Santa Barbara, CA, November 1992.
5. Porras, P. "STAT, A State Transition Analysis Tool for Intrusion Detection." Master thesis, Computer Science Department, University of California, Santa Barbara, CA, July 1992.
6. Kumar, S. and E. H. Spafford, "A Pattern Matching Model for Misuse Intrusion Detection." Proceedings of the Seventeenth National Computer Security Conference, October 1994: 11-21.
7. Mounji, A. "Languages and Tools for Rule-Based Distributed Intrusion Detection." Thesis, Faculte's Universitaires Notre-Dame de la Paix Namur, Belgium, September 1997.

8. Smaha, S. and S. Snapp. "Method and System for Detecting Intrusion into and Misuse of a Data Processing System." US555742, U.S. Patent Office, September 17, 1996.

9. `http://www.nai.com/asp_set/products/tns/ccmonitor_intro.asp`.

10. `http://www.nfr.net`.

11. Anderson, R. and A. Khattak. "The Use of Information Retrieval Techniques for Intrusion Detection." Presentation at the First International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium, September 1998.

12. Denning, D. "An Intrusion Detection Model." Proceedings of the Seventh IEEE Symposium on Security and Privacy, May 1986: 119–131.

13. Hochberg, J., K. Jackson, C. Stallings, J. McClary, D. DuBois, and J. Ford. "NADIR: An Automated System for Detecting Network Intrusion and Misuse." *Computers and Security*, vol. 12, Elsevier Science Publishers, Ltd., 1993: 235–248.

14. Javitz, H. S. and A. Valdes. "The SRI IDES Statistical Anomaly Detector." Proceedings IEEE Symposium on Security and Privacy, Oakland, CA, May 1991.

15. Smaha, S. E. "Haystack: An Intrusion Detection System." Proceedings of the Fourth IEEE Aerospace Computer Security Applications Conference, Orlando, FL, December 1988.

16. Mukherjee, B., L. T. Heberlein, and K. N. Levitt. "Network Intrusion Detection." *IEEE Network*, vol. 8, no. 3, May/June 1994: 26–41.

17. Hochberg, J. et al., op cit.

18. Lankewicz, L. and M. Benard. "Real Time Anomaly Detection Using a Nonparametric Pattern Recognition Approach." Proceedings of the Seventh Annual Computer Security Applications Conference, San Antonio, TX, December 1991.

19. Liepins, G. E. and H. S. Vaccaro. "Intrusion Detection: Its Role and Validation." *Computers and Security*, vol. 11, Elsevier Science Publishers, Ltd., 1992: 347–355.

20. Teng, H. S., K. Chen, and S. Lu. "Adaptive Real-Time Anomaly Detection Using Inductively Generated Sequential Patterns." Proceedings of the IEEE Symposium on Security and Privacy, May 1990.

21. Debar, H., M. Becker, and D. Siboni. "A Neural Network Component for an Intrusion Detection System." Proceedings of the 1992 IEEE Symposium on Security and Privacy: 240–250.

22. Hofmeyr, S. A, S. Forrest, and A. Somayaji. "Intrusion Detection Using Sequences of System Calls." *Journal of Computer Security*, vol. 6, no. 3, 1996.

23. Warrender, C., S. Forrest, and B. Pearlmutter. "Detecting Intrusions Using System Calls: Alternative Data Models." Proceedings of the Twenty-Fifth IEEE Symposium on Security and Privacy, Oakland, CA, May 1999.

24. Howe, D. Free On-line Dictionary of Computing (FOLDOC), available at `http://foldoc.doc.ic.ac.uk`.

25. Mé, L. "GASSATA, A Genetic Algorithm as an Alternative Tool for Security Audit Trails Analysis." First International Workshop on the Recent Advances in Intrusion Detection, Louvain-la-Neuve, Belgium, September 1998.

26. Balasubramaniyan, J. S., J. O. Garcia-Fernandez, D. Isacoff, E. H. Spafford, and D. Zamboni. "An Architecture for Intrusion Detection Using Autonomous Agents." COAST Technical Report 98/05, Purdue University, June 1998.

27. Porras, P. A. and P. G. Neumann. "Emerald: Event Monitoring Enabling Responses to Anomalous Live Disturbances." Proceedings of the Twentieth National Information System Security Conference, Baltimore, MD, 1997.

28. Neumann, P. G. and P. A. Porras. "Experience with EMERALD to Date." First USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, CA, April 1999.

29. Lee, W., S. J. Stolfo, and K. W. Mok. "A Data Mining Framework for Building Intrusion Detection Models." Proceedings of the Twentieth IEEE Symposium on Security and Privacy, Oakland, CA, 1999.

CHAPTER 5

Responses

After the analysis is done, and the system has identified problems, it's time to let someone know about them (and in some cases, to take additional action). In the intrusion detection process model, this is handled by the *response* section. Ideally, this portion of the system is feature-rich and serves all the members of the security management team by tailoring responses to each of them.

This chapter covers numerous ways in which intrusion detection systems can handle the results of analysis and outlines some options for responses to detected problems. These options include passive responses, in which the system simply notes and reports the problem; and active responses, in which the system (automatically or in concert with the user) takes action in order to block or otherwise affect the progress of the attack). Finally, we discuss ways of tying the results back into the site security management process.

5.1 Requirements for Responses

Many considerations come into play when designing response features for an intrusion detection system. Some responses can be designed to reflect current standards for security management or incident handling; others can reflect local management concerns and policies. When designing response features for commercial products, vendors should provide end users with the capability to tailor response mechanisms to fit their particular environment.

In the early days of intrusion detection system research and design, most designers focused on the monitoring and analysis sections of the system, leaving the crafting of the response component to the user. Although there was a great deal of discussion of what users really wanted in a response component, no one had a clear idea of the operational environment in which intrusion detection systems were likely to be fielded. In one of the first intrusion detection research conferences, an intrusion detection system researcher presented his tongue-in-cheek design for "the perfect intrusion detection system" (pictured in Figure 5.1). It remains one of my favorite designs, because it so accurately captures my experience with early users of the technology!

Note

A question that arises from time to time has to do with the generic term *user*. Who is the "standard user" of an intrusion detection system?

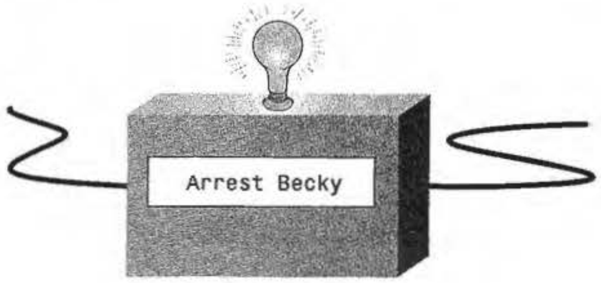
I've found three general categories of users for these systems in my experience so far. First are network security specialists or managers. Specialists are sometimes brought in as consultants to the systems management staff. Because these security specialists interact with various commercial intrusion detection systems, these people are often very knowledgeable about the various tools. However, the specialists are not always as knowledgeable about the underlying systems that they are monitoring or testing.

Second are the system administrators who are using the intrusion detection products to monitor and protect the systems they manage. In some cases, these are the power users of the intrusion detectors, with a good technical understanding of both the tools and the system environments they protect. System administrators can be the most demanding users of intrusion detection systems, sometimes requiring features seldom used by the other user constituencies of the products.

Finally, we have security investigators, members of a system audit staff or a law enforcement agency, who use the intrusion detection product to monitor compliance with legal regulations or to support an investigation. These users may not possess the technical acumen to understand either the tools or the underlying systems. However, they do understand the process of investigating problems and can be the source of important insight for intrusion detection system designers.

In this chapter, we will refer to these three groups of users as *security managers*, *systems administrators*, and *investigators*, respectively. The term "user" will refer to all three groups.

Figure 5.1 The Perfect Intrusion Detection System[1]



Users rely on intrusion detection systems to perform complex and exacting analysis on huge volumes of system event data. Ultimately, they want the system to be reliable and accurate and to convey the results of the analysis to the right people at the right time in straightforward, easy to understand terms. Many considerations color these requirements, but the goal remains the same.

5.1.1 Operational Environment

When designing a response mechanismm, an obvious consideration is the nature of the operational environment in which the intrusion detection system is operating. The alarm and notification requirements of an intrusion detection system that has a number of attended control consoles lining the wall of a network operations center are likely to be quite different from the requirements of an intrusion detection system installed on a desktop system in a home-based business.

The information provided by the intrusion detection system as part of the notification also depends on the environment. Network operations center staffers might prefer to use products that provide details about low-level network traffic (contents of fragmented packets, for instance). A graveyard-shift security manager might consider anything beyond a simple alarm with a message to contact the proper person to be worthless.

Audible alarms are perfectly suitable for installations in which one person is responsible for monitoring the results of multiple intrusion detection systems. Such alarms can be a massive annoyance for those managing multiple operations on a complex network from a single console.

Visual alarms and activity graphs may be of value to installations that have a full-time operator who sits in front of the system console. They are especially helpful when monitoring other components of the security infrastructure (such as encryptors or firewalls) that might not be visible from the management area.

Visual alarms are likely to be irrelevant to operators who are not present to view them. Color-coded alarm status displays are of little value to operators who are color-blind (as are a significant percentage of military system operators) or to those who are visually challenged.

5.1.2 System Purpose and Priorities

Another factor driving response requirements is the monitored system's function. The need to provide active responses (for instance, terminating the network connections of a user recognized as the source of an attack) is driven in part by systems that provide critical data and services to users. An example of this type of system is a medical record server for an emergency room. Another is a Web server for a high-traffic, high-revenue electronic commerce site.

In these cases, the impact of a successful denial-of-service attack (or a series of such attacks) can be devastating. In both cases, the value of preserving the availability of the systems far outweighs any additional overhead associated with providing active responses to detected intrusions.

5.1.3 Regulatory or Statutory Requirements

Other factors driving specific response capabilities include regulatory or statutory requirements for intrusion detection. In certain military computing environments, intrusion detection capabilities are required for certain types of processing to take place. For instance, a system might be accredited to handle classified information of a certain level of sensitivity only if an intrusion detection system is present. In these environments, regulations govern the operation of the intrusion detection systems, and reporting requirements govern the format and delivery schedule for the results of the intrusion detection system's operation. If the intrusion detection system is not running, the regulations dictate that the classified information cannot be processed on the system.

In online stock-trading environments, the Securities and Exchange Commission requires trading systems to be accessible to customers during trading hours. Any denial of access can result in fines and other penalties levied on the site. This situation requires both automated responses that can block attacks and well-targeted, concise explanations of detected problems—so that damage recovery can be completed as quickly as possible.

5.1.4 Conveying Expertise to Users

One need that intrusion detection products often neglect is providing guidance to users along with or as part of the detection responses. In other words, whenever possible, the system should accompany the detection results with explanations and advice to the user that allow him to take appropriate action. This area is one in which an immense disparity exists among products. A well-designed set of response mechanisms can structure information and explanations so that users are guided through a series of decisions, in the proper order, that ultimately lead them to the appropriate resolution of the problem.

Such a response mechanism also allows the tailoring of the presentation of results to users possessing different levels of expertise. As noted in the description of user constituencies in the author's note, different users of intrusion detection systems have different information needs. System administrators are probably able to make sense of sequences of network service requests or raw packet traces. Security specialists might be able to understand the difference between a "port scanner" and a "sendmail buffer overflow." Investigators might require the capability to track the sequence of commands a particular user makes, along with the system objects that user affects.

Intrusion detection system developers should accommodate a wide range of user capabilities and expertise levels. In this fast-growing market (that did not exist 10 years ago) the expert user is likely to be rare for a few more years!

5.2 Types of Responses

Intrusion detection system responses can be classified as *active* or *passive*. In active responses, the system (automatically or in concert with the user) blocks or otherwise affects the progress of the attack. In passive responses, the system simply reports and records the problem.

Active and passive responses are not mutually exclusive. Intrusion detection systems should, as a matter of course, always log detection results, regardless of whether other responses are enabled.

An essential part of including intrusion detection in a site security process is determining which intrusion detection responses to enable and deciding which actions should occur as a result of those responses.

5.2.1 Active Responses

As stated above, responses can be classified as active or passive. Active responses involve taking action based on the detection of an intrusion. Several options are available for active responses; most of these options fall into one of the following categories:

- Take action against the intruder
- Amend the environment
- Collect more information

Although the first option, taking action against the intruder, is extremely popular in some circles, it is not the only active response. Furthermore, because it has serious legal and practical implications, this response should not be the most common active response you use!

There are two forms of active response: those that are user driven and those that are performed automatically by the system.

5.2.1.1 Take Action Against the Intruder

The first option in active response is to take action against intruders. The most aggressive form of this option, tracing back the intruder to the source of the attack and then taking action to disable the intruder's machine or network connection, has captured the imagination of many an information warfare groupie. This approach is also of considerable appeal to the long-suffering security manager who has been the target of one too many hacker denial-of service-attacks!

Unfortunately, this option can also represent one of the biggest briar patches of security. The hazards represented by striking back include the following:

- Given the general *modus operandi* of network hackers, the system identified as the source of the attack on your system probably belongs to another victim of the hacker. Network hopping, in which a hacker successively hacks a system and then uses it as a platform for attacking another system, is a common practice. If you target the system from which the attack was launched, you are probably targeting an innocent party.
- Even if the attacker is coming from a system over which she or he has legitimate control, spoofing the IP address of the source of attacks is common practice. Therefore, the IP address that appears to be the source of the attack on your system may actually be "borrowed" from another (innocent) victim.
- You may find that striking back simply provokes an escalation of the attack. What might have begun as a routine surveillance or scan of your system could develop into a full-scaled hostile attack, placing the availability of your system resources in jeopardy.
- In many situations, by striking back you expose yourself to a significant risk of criminal charges or civil legal action. If your actions attack an innocent party, that party may sue you for damages. Furthermore, your reaction in itself may violate computer crime statutes, and you may be subject to charges. Finally, if you work for a government or military organization, you may be violating policy and may be subject to disciplinary action or dismissal. Law enforcement officials advise contacting authorities for assistance in dealing with attackers.[2]

Taking action against intruders can also occur in more benign forms. For instance, the intrusion detection system might simply terminate the network session by resetting TCP connections. The system might also direct a firewall or router to block packets coming from the IP address that appears to be the source of the intrusion.

Another response is to automatically spawn email to the administrator of the system from which the intruder appears to be coming and request assistance in identifying and dealing with the problem. This can be productive when the hacker is connected to that system by a dial-up connection. As traceback capabilities improve for the communication infrastructure as a whole, it may become possible to utilize features in the telephone system (such as caller ID or trap and trace) to assist in establishing accountability for intrusions.

User-Driven Responses

Many active response capabilities originated in the days when "super security geeks" performed them manually. Although many of the responses can be automated to deal with attacks in real time, this doesn't mean you should.

For instance, suppose an attacker discovers that your system's automated response to a denial-of-service attack is to "shun" (that is, terminate the current connection and refuse subsequent TCP connections with the source IP address) the ostensible source of the attack. The attacker may use IP-spoofing tools to generate denial-of-service attacks that appear to come from a list of your most important customers, resulting in those customers being denied access to your critical resources. What is even worse is that in strictest terms, this denial of service is being enabled by your own intrusion detection system!

Automatic Responses

On the other hand, automating at least some of the active responses is necessary because of the sheer speed with which attacks take place. Most of the attacks that come from the Internet utilize attack software and scripts. These attacks progress at a pace that prohibits manual intervention. Intrusion detection designers should consider whether a particular active response can be handled manually (with the intrusion detection system providing guidance and information to the user). If the intervention must be automated, measures should be taken to minimize the risk of the automated response being used as a vehicle for attack. We will return to this topic in Chapter 12, "For Designers," when we cover design issues for intrusion detection.

5.2.1.2 Amend the Environment

The next option for active response is to amend the system environment. Although this type of response to intrusions is quieter and less glamorous than other approaches, it is often the optimal response scheme, especially in combination with responses that provide investigative support. The concept of amending the system environment to "heal" flaws that allow intrusions to occur is consistent with the vision for critical systems articulated by many researchers. "Self-healing" systems are equipped with defenses analogous to the body's immune system in which problems can be recognized, the causative factors isolated, and a suitable response generated to address the problem.

In some intrusion detection systems, this category of response could alter the operational characteristics of the analysis engine, perhaps increasing sensitivity levels. It could also alter expert systems by inserting rules that increase the suspicion level for certain attacks or increase the scope of the monitoring to collect information at a finer granularity than usual. This strategy is analogous to those used in real-time process control systems in which the outcome of the current system process is used to tune and refine subsequent processes.

5.2.1.3 Collect Additional Information

The third option in active response is to collect additional information. This option is of special interest when the system being protected is critical and a system owner might want to pursue legal remedies. At times, this logging response is coupled with the use of a specialized server, established to serve as an environment into which intruders can be diverted. This server is known by a variety of names. Most common are "honey pots," "decoys," or "fishbowls." Such servers are equipped with file systems and other spoofed system attributes that are designed to mimic the appearance and content of critical systems.

Note

The design of decoy servers was first explored in 1992, in Bill Cheswick's classic paper, "An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied."[2] This paper outlines the steps that Cheswick took to design, build, and field a decoy server into which he redirected a Dutch hacker who attacked Cheswick's systems. The use of a decoy server was reported by Cliff Stoll in his classic book *The Cuckoo's Egg*.[3]

Decoy servers are of value to security managers who are collecting threat information on intruders or who are collecting evidence to support taking legal action against them. Using a decoy server allows the victim of an intrusion to determine the intent of the intruder, logging extensive information about the activities of the intruder without placing the actual system contents at risk of damage or divulgence. This information can also be used to construct custom detection signatures.

Information collected in this way is also of value to those performing trend analysis of network security threats. This information is of particular interest in systems that must operate in hostile threat environments or that are subject to large numbers of attacks (such as government Web servers or high-profile electronic commerce sites).

5.2.2 Passive Responses

Passive responses are those that provide information to the user, relying on the user to take subsequent action. In early intrusion detection systems, all responses were passive. Passive responses are important, however, and in many cases represent the sole response form for the system. In this section passive responses are presented in order of criticality to the user. (This criticality is the primary difference between alarm mechanisms and problem reports.)

5.2.2.1 Alarms and Notification

Most intrusion detection systems provide options for generating alarms in a variety of forms. This flexibility allows a user to tailor the alarms to fit the organization's system operating procedures.

Alarm Display Screens

The most common alarm and notification feature provided by intrusion detection systems is an onscreen alert or window. This alarm and message appear on the intrusion detection system console or on other systems as specified by the user in the intrusion detection system setup. Different systems provide different levels of detail in the alarm message, ranging from a simple "an intrusion has occurred" to extensive records outlining the ostensible source of the problem, the target of the attack, the apparent nature of the intrusion, and whether it was successful. In some systems the contents of the alarm message can also be customized.

Remote Notification of Alarms and Alerts

Organizations that run attended systems around the clock use another alarm/alert option. In these situations, intrusion detection systems can issue alarms and alert messages by dialing pagers or cellular telephones issued to system administrators and security personnel. Email messages are another means of notification, although this approach is not recommended in cases of on-going or persistent attacks (the attacker is likely to read or, worse, block the email message). In some cases, the notification option allows users to configure additional information or alarm codes sent to these units.

5.2.2.2 SNMP Traps and Plug-Ins

Some intrusion detection systems are designed to function in concert with network management tools. These systems can utilize the network management infrastructure to send and display alarms and alerts on the network management console. Some products spawn Simple Network Management Protocol (SNMP) messages or traps as an alarm option.

This option is currently provided in some commercial products, but many believe that intrusion detection and network management systems can be much more thoroughly integrated. Several benefits are associated with this integration, including the ability to utilize common communications channels and the ability to provide active responses to security problems that take into consideration the network environment at that time. Furthermore, SNMP traps allow users to move the processing load associated with responding to a detected problem to the system receiving (and acting upon) the trap.

5.3 Covering Tracks During Investigation

Part of the effectiveness of an intrusion detection system relies on its capability to provide silent, reliable monitoring of attackers. When a system is under attack, it is wise to handle alarms and notification in a fashion that is invisible to the intruder. This approach allows investigative activity to take place while the intruder's session is still underway, allowing accountability to be established. In these cases, alarms and notification may be performed over encrypted channels. Other needs for encrypted channels are discussed in the following section on fail-safe considerations.

5.3.1 Fail-Safe Considerations for Response Components

Several fail-safe measures should be taken in this component of intrusion detection systems.

First, as in the rest of the system, the design of the response system and all of its components should assume that an adversary will target them as part of the attack. The attack strategy is likely to involve either monitoring the response channels, searching for signs of detection, or else disrupting or intercepting the alarm and alert channels so that operators are not notified of the attack.

Utilizing encrypted tunnels and other cryptographic means of hiding and authenticating intrusion detection communications is required for the reliable operation of the system. This measure precludes a variety of attacks targeting both the response components and the rest of the intrusion detection system.

Alarms generated by intrusion detection systems should be redundant, utilizing multiple channels. For example, users might want to configure an intrusion detection system to trigger three alarms for a certain severity of attack on a critical system, sending one alarm to the notification unit via normal network communications, a second via encrypted channels, and a third via a dial-up channel.

The logs generated by the response component that document all detection results should be protected from alteration or destruction. Because they are likely to be used in any investigation and to support any legal action, this protection is especially important. One way to protect these logs is to use write-once media (such as CD-ROMs) with an optional hard copy backup to line printers. For especially critical systems, redundant logging mechanisms are recommended.

5.3.2 Handling False Alarms

One problem that exists in intrusion detection systems and that requires some embedded intelligence in the response component of the system involves false alarms. These alarms can be false positives, in which the system identifies attacks when there are none, and false negatives, in which the system fails to identify attacks when they occur. False negatives are a problem of analysis and are best handled in that component. However, false positives can in themselves present problems to the response component.

When a faulty network component corrupts packets, triggering a false indication of network attack, the analysis system sends an alarm to the response component, which will in turn generate alarm messages. If the corrupted packets trigger false alarms at even two per second, the response component may be flooded with service requests and ultimately crash. Therefore, response components should give users the capability to limit the number of alarms triggered by the same source and signature during a time interval. This technique not only serves to protect the integrity of the response component but also reduces the possibility that the user will ignore the alarms generated by the detector.

5.3.3 Archive and Report

The longer-range portion of passive response involves archiving detection results for later use. Some intrusion detection systems store the results in databases. This approach allows the user to generate a wide range of reports, targeting each to a particular audience. This feature of intrusion detection products is very popular because it allows security managers to regularly inform executive management about the state of system security, targeting details of problems to those best equipped to deal with them.

Another task that is important to the security process is to maintain an intrusion detection result log, structured in much the same way as system logs and audit trails. This log should be written to write-once media to protect it from alteration or deletion. This log is important because it provides a long-term sequential record of intrusions against the target system. This material can be important as documentation of the progress of a long-term problem and can, even more importantly, serve as evidence should the organization decide to seek legal remedies for the problem. Such evidence is critical regardless of whether the remedies pursued are in criminal or civil venues.

5.4 Mapping Responses to Policy

A successful security management program effectively blends policy and supporting technology. To optimize the utility of intrusion detection systems, it should be included in organizational security policy and procedures. One way to do this is to include provisions to specify which activities should correspond to detected intrusions or security violations. These activities are divided into the following four categories, ordered by the time and criticality of the activities: immediate or critical, timely, local long-term, and global long-term.

5.4.1 *Immediate*

Immediate or critical actions are those required of system management immediately following an intrusion or attack. These include the following:

- Initiating incident-handling procedures
- Performing damage control and containment
- Notifying law enforcement or other organizations
- Restoring victim systems to service

The time span associated with immediate or critical action can be determined by local policy and can be further refined to accommodate the severity of the attack.

5.4.2 *Timely*

Timely actions are those required of system security management following detected attacks or security violations. The elapsed time can range from hours to days, and these actions usually follow those in the immediate/critical category. Activities that should take place in a timely fashion include the following:

- Manually investigating unusual patterns of system use
- Investigating and isolating the root causes of the detected problems
- Correcting these problems when possible (by applying vendor bug patches or reconfiguring systems)
- Reporting the details of the incident to proper authorities (if they were not involved in the incident-handling process)
- Altering or amending detection signatures in the intrusion detection system
- Instigating or pursuing legal action against the perpetrator
- Dealing with publicity associated with the attack and notifying shareholders, regulators, and others for whom there may be statutory reporting requirements

5.4.3 *Long-Term—Local*

Local long-term actions refer to system management activities that are less critical than actions that fall in the immediate or timely categories but are still important to the security management process. The impact of these activities is local to the organization. These activities might be scheduled as part of a regular review.

Activities that fall into this category include the following:

- Compiling statistics and performing trend analysis
- Tracking patterns of intrusion over time

These patterns of intrusion and security violation should be evaluated to isolate areas requiring amendment or improvement. For instance, a large number of attacks targeting a vulnerability that has been corrected by a vendor might lead to a security policy requirement that systems software be patched on a regular schedule. A large number of false alarms might indicate a need to review the detection signatures or configuration of the intrusion detection system or else to look at an alternative intrusion detection product. Finally, large numbers of problems that are due to user error might indicate a need for additional training of personnel.

5.4.4 Long-Term—Global

Global long-term actions refer to system management activities that are noncritical but nonetheless important to the state of security on a societal level. The impact of these activities is not confined to the organization. These activities are likely to be conducted in the context of an industry organization or consortium.

Activities in this category include the following:

- Notifying vendors of the problems the organization has suffered due to security problems in their products
- Lobbying lawmakers and the government for additional legal remedies to system security threats
- Reporting statistics regarding security incidents to law enforcement or other organizations that maintain statistics

Many critical issues in system security simply can't be addressed at a local level. Thus community-level activity allows you and your organization to be part of the larger remedy.

5.5 Conclusion

In this chapter we covered the third and final component of the intrusion detection process model: response. This component handles the output of the analysis component, generating both active and passive responses to intrusions that are detected.

We explored requirement definitions for this functional component and defined a classification scheme for binding the results of the intrusion detection process to the organizational security management process.

Endnotes

1. Smaha, Steve. Presentation, First Experts Conference on Future Directions in Computer Misuse and Anomaly Detection, Davis, CA, April 1992.
2. Yasin, R. "Think Twice Before Becoming a Hacker Attacker." *Internet Week*, December 14, 1998.
3. Cheswick, W. "An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied." Proceedings of the USENIX Security Conference, San Francisco, CA, Winter 1992: 163–174.
4. Stoll, Clifford. *The Cuckoo's Egg: Tracking A Spy Through the Maze of Computer Espionage.* New York, NY: Doubleday, 1989.

TECHNOLOGY SERIES

INTRUSION DETECTION

Rebecca Gurley Bace is the President of Infidel, Inc., a consulting practice specializing in intrusion detection and network security technology and strategy. Prior to founding Infidel, Ms. Bace spent thirteen years in government, the first twelve as an employee of the National Security Agency. She led the Computer Misuse and Anomaly Detection (CMAD) Research program from 1989 through 1995, as a charter member of NSA's Office of Information Security Research and Technology (R2). As the leader of CMAD research, Ms. Bace championed much of the early research in Intrusion Detection, sponsoring academic research at Purdue University (COAST project), the University of California, Davis (Security Lab), University of New Mexico, and Tulane University. Ms. Bace's research collaborations with Dr. David Icove of the Federal Bureau of Investigation led to the commercial publication of a manual for computer crime investigation. She and the CMAD workshop series she founded and sponsored were involved in the 1995 detection, traceback, and apprehension of Kevin Mitnick, at the time the FBI's most wanted computer criminal. Ms. Bace received a NSA Distinguished Leadership Award in 1995, in recognition of her work building the national CMAD community. After leaving the NSA in 1996, Ms. Bace served as the Deputy Security Officer for the Computing, Information, and Communications Division of the Los Alamos National Laboratory. A native of Leeds, Alabama, Ms. Bace holds the Bachelor of Science from the University of the State of New York, and the Master of Engineering Science from Loyola College.

The *Technology Series* is a comprehensive and authoritative set of guides to the most important computing standards of today. Each title in this series is aimed at bringing computing professionals closer to the scientists and engineers behind the technological implementations that will change tomorrow's innovations in computing.

With the number of intrusion and hacking incidents around the world on the rise, the importance of having dependable intrusion detection systems in place is greater than ever. Offering both a developmental and technical perspective on this crucial element of network security, *Intrusion Detection* covers:

- Practical considerations for selecting and implementing intrusion detection systems
- Methods for handling the results of analysis, and the options for responses to detected problems
- Data sources commonly used in intrusion detection and how they influence the capabilities of all intrusion detection systems
- Legal issues surrounding detection and monitoring that affect the design, development, and operation of intrusion detection systems

More than just an overview of the technology, *Intrusion Detection* presents real analysis schemes and responses, as well as a detailed discussion of the vulnerabilities inherent in many systems, and approaches to testing systems for these problems. Ideal for the network architect who has to make decisions on what intrusion detection system to implement and how to do it, this book will help you:

- Understand the history of the technology, as well as how future changes may affect your systems
- Guide an organization through a full acquisition lifecycle, from initial requirements definition to product deployment
- Choose your systems' responses to detected problems and tie the results back into the site security management process
- Assess the quality of a proposed or existing intrusion detection system design

New Riders

CATEGORY: Networking
www.newriders.com
$50.00 USA / $74.95 CAN

ISBN 1-57870-185-6
95000
6 19472 70185 0
9 781578 701858