HOST-BASED MISUSE DETECTION AND CONVENTIONAL OPERATING SYSTEMS' AUDIT DATA COLLECTION

A Thesis

Submitted to the Faculty

of

Purdue University

by

Katherine E. Price

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

December 1997

To my parents, Rick and Pat, and my fiancé, Dan.

ACKNOWLEDGMENTS

Many people have made contributions to the development of this work. First off, I thank Dr. Spafford for his guidance. The process leading to this work was long and at times seemed impossible, but Spaf was always there to bring me back to reality when I wandered too far astray. I also thank my other committee members, Dr. Mike Atallah and Dr. Carla Brodley, for their time and energy, especially when it came to scheduling a defense, a task that proved to be almost as difficult as compiling this work. I also thank Dr. Gorman for his feedback and his words of wisdom, even though sometimes I foolishly did not heed them.

I thank all my past and present friends at Purdue for their friendship and support. Many special thanks to everyone in COAST who never let me take life too seriously. I especially acknowledge the tremendous task Marlene Walls has tackled in the organization of Spaf, and I thank her for always squeezing me into his schedule with a smile.

I thank my parents, Pat and Rick Price, for supporting me throughout my educational career. Their support opened doors for me that may have otherwise been closed. Also, I thank my sister Becky Price for always making my days look simple and my problems surmountable when compared to her whirlwind tour of life.

I thank Boo Bear and Oscar for always being there purring for me, no matter how foul my mood when returning from work. And finally, I thank Dan Schikore for keeping me sane throughout the development of this work.

Portions of this work were supported by contract F30602-96-2-0268 from Rome Labs (USAF), and by the sponsors of the COAST Laboratory.

DISCARD THIS PAGE

TABLE OF CONTENTS

Page

DISCARD THIS PAGE

LIST OF TABLES

DISCARD THIS PAGE

LIST OF FIGURES

ABSTRACT

Price, Katherine. M.S., Purdue University, December 1997. Host-Based Misuse Detection and Conventional Operating Systems' Audit Data Collection. Major Professor: Eugene H. Spafford.

Computing systems have evolved from stand-alone mainframes to complex, interconnected open systems, and this evolution has led to a proliferation of avenues of attack. With the knowledge that system misusers have open avenues for attack, misuse detection provides an important line of defense. For a misuse detection system to be effective, there needs to be an audit trail of system activity that was designed to support misuse detection needs.

A major challenge in misuse detection is that audit data is inadequate. The data supplied by current auditing systems lack content useful for misuse detection, and there is no widely accepted audit trail standard.

This thesis presents a comparison of the needs of host-based misuse detection with the capabilities of auditing facilities of conventional operating systems. Host-based misuse detection systems are examined, and the audit data used by each are outlined. Auditing systems of conventional operating systems are also examined, and the data collected by each are outlined. A comparison of the needs of the misuse detection systems and the capabilities of existing auditing facilities is then presented. The results of this study aid in the determination of what data content should be provided by auditing systems for the support of misuse detection goals.

1. INTRODUCTION

In this chapter, we review the rise of computer crime and discuss the importance of computer security. We motivate the need for misuse detection and discuss misuse detection's reliance on audit trails. Finally, we will present the problems we explore in this thesis and outline the remaining chapters of this work.

1.1 Computer Crime

The communications revolution created by the integration of computing systems and networks into almost every aspect of society has affected business, education, travel, medicine, and research. These new technologies are enabling changes in how people work and live [III93]. The communications technologies are making possible collaborations and relationships that were previously restricted by geographic separation. Computers are taking over routine work and creating entirely new types of jobs such as system controller and computer engineer. Computers and networks are increasingly allowing people to live and work where and how they wish. Unfortunately, the communications revolution is also enabling a new wave of crime.

Computer crime is on the rise, and estimates for losses from computer fraud are in the billions of US dollars. The 1997 CSI/FBI Computer Crime Survey [Pow97] revealed that 75% of the surveyed organizations reported financial losses from computer security breaches. The damage to the 249 organizations who were able to quantify their losses totaled over $100,000,000.

Sensitive information that previously were secured in file cabinets are now being stored and transmitted using computer systems and global networks. Millions of financial transactions are transmitted over U.S. networks every day. In the 1996 *Information Week*/Ernst & Young Information Security Survey [Vio96], one-third of the respondents reported using the Internet for exchange of business correspondence. The people moving the information to computer systems have often failed to consider the security requirements necessary to safeguard their assets, and so are providing new targets for theft, fraud, vandalism, and espionage [ISV95].

The evolution from stand-alone machines to complex, interconnected systems has led to computer systems that are increasingly vulnerable to computer attacks [SSTG92, Pro94, ISV95, Pre97]. Most existing system have known security flaws and cannot be replaced for economic reasons. Additionally, finding and fixing all the security weaknesses is often technically infeasible. Increasingly, organizations are relying on unsecure computer systems, and, increasingly, criminals are finding fewer obstacles exist to computer crime than conventional means.

In addition to organizations' reliance on unsecure systems, few computer attacks are ever detected, and even fewer are reported. Fears of loss of reputation, consumer confidence, and competitive edge have lead to organizational aversions to reporting computer crime [ISV95, Pow97]. Finally, computer crime is often difficult to prosecute. Criminal law has not kept up with technology changes, and so computer incidents are rarely treated with the severity accorded to conventional crimes [ISV95].

With knowledge that computer systems will come under attack, organizations must assess their threats and develop defense strategies to safeguard their assets and resources.

1.2 Computer Security

No organization has unlimited time and resources to devote to computer security. Thus the first step to improving a system's security is to perform a risk analysis. A *risk analysis* is the process of identifying a system's threats and vulnerabilities, and the countermeasures that can be taken to protect the system [ISV95]. A risk analysis helps an organization identify and weigh their risks and determine reasonable costs to reduce the risks to acceptable levels.

1.2.1 Risks

Many risks to both tangible resources, such as money or proprietary information, and intangible resources, such as reputation, consumer confidence, or competitive advantage, face an organization [GS96]. Loss or damage to any of these resources could be disastrous for an organization. The risks facing an item are often divided into three basic categories:

- *Loss of Confidentiality* – Confidentiality may be lost through unauthorized access, disclosure, observation, or copying.
- *Loss of Integrity* – Integrity may be lost through unauthorized modification, repudiation, false data entry, or misrepresentation.
- *Loss of Availability* – Availability may be lost through unauthorized destruction, contamination, or monopolization.

During the risk analysis, all items considered of value are identified, and the effects of each item's loss or damage are determined in terms of lost money or time. Once all items of value are identified with their risks, threats to the items are determined.

1.2.2 Threats

Any possible danger to the system constitutes a threat [ISV95]. A threat might be a person, such as a computer criminal; a thing, such as faulty software; or an event, such as a power outage. During the risk analysis, possible threats to an organization's resources are identified. Threats are often characterized based on whether they are internal or external to the organization and whether they are malicious or accidental in nature.

1.2.2.1 Insider vs. Outsider Threats

A threat might arise from outside or from within an organization. Historically, the vast majority of computer crime was committed by people from within the organization. Insiders still account for the most theft of proprietary information, disruption of service, and loss of confidentiality, but recently the threat from outside has been rising dramatically [Pow96, Pow97]. Though the threat from outside may never surpass the insider threat, it may soon match it. The rise in the threat from outside is possibly because of the similarly dramatic increase in network connectivity.

Threats from individuals with legitimate access to the system are of particular concern because so many people are in this category. For example, employees, contractors, and client representatives working on site have authorized access to the computer system, and, even if an individual is not an authorized computer user, physical access to the equipment can be abused. Insider threats are not limited to people with technical savvy. Simply copying a confidential file onto a diskette and dropping the diskette in the mail can result in major losses for the organization.

1.2.2.2 Malicious vs. Accidental Threats

Threats fall into two main categories: accidental and malicious. Accidental threats include natural and physical dangers, such as floods, power failures, and earthquakes, and unintentional dangers, such as careless users inadvertently erasing files. Malicious threats include intentional dangers, such as computer crime and industrial espionage.

The term *intruder* is often used to describe a malicious threat. The teenager hacking into systems for fun and thrill is the type of intruder that has received recognition in the popular media. Unfortunately, many organizations are faced with graver threats. The Intruder Classification Model [And94] classifies intruders into four categories: individual intruders, organized groups, criminals, and espionage agents. The categories, which are differentiated by motivation, provide a better understanding of why computer intrusions are taking place.

Individual intruders are motivated by the challenge and thrill of gaining access to a computer system, and normally perform little or no strategic planning. Organized groups are motivated by gain of specific information or system resources to achieve common goals, and the members often cooperate to define strategies, and to select and research targets. Criminals are motivated by profit or unfair market share, and weigh the cost of the computer crime against more traditional methods. Espionage agents are motivated by national economic or strategic objectives, and exhibit the greatest variety and complexity of methods and resources.

Note that not all human threats are maliciously motivated. People make mistakes, especially if untrained or careless. Though a mistake may be accidental in nature, it can still be damaging. An accidental release of confidential information may be as damaging to an organization as if it was disclosed through a criminal attack. Thus, accidental threats are as big of a concern as malicious threats.

1.2.3 Vulnerabilities

Once the threats are identified, the vulnerabilities that may be exploited by the threats are examined. A vulnerability is a point where the system is susceptible to attack [ISV95]. A vulnerability might be a person, such as a system operator; a thing, such as a network connection; or a system characteristic, such as the fact that the facility is located in a flood zone. The risk analysis identifies the system's vulnerabilities that may be exploited by a threat.

Many factors have contributed to creating an environment that fosters the development of vulnerabilities in computing systems [Den87, And94, GS96]:

- *Flawed system components* – System components are often designed and implemented without security as a priority. Additionally, developing absolutely secure systems is generally impossible. Security weaknesses in components and lack of a security focus leads to vulnerabilities in integrated computing systems.

- *Geographical distribution* – Computer systems are located on every continent and are connected into a global network. These systems are logically accessible from anywhere in the world making geographic distribution no longer an obstacle to access. Geographical distribution of systems makes control of physical and logical access to each system difficult.

- *Size and complexity* – Computing systems often support thousands of users and are composed of hundreds of individual computers connected into a local area network. Additionally, these systems are interconnected by the thousands to form wide area networks. Lack of a full understanding of the underlying systems, network topologies, and multiple points of access allows vulnerabilities to develop unnoticed.

- *Frequency of change* – The speed and frequency of change in technologies and applications of computer systems grow with the systems' size and complexity. Systems, applications, and network connections are added and removed daily. The frequency of change in computing systems leads to problems in tracking and controlling the systems, thus allowing vulnerabilities to develop.

Vulnerabilities exist and sometimes are inherent in today's computer systems. Finding and correcting all the vulnerabilities is often infeasible for economic and technical reasons, and the replacement of many of the systems with more secure systems is economically impractical. Also, absolutely secure systems are difficult, if not generally impossible, to develop, and secure systems are still vulnerable to abuses by insiders with legitimate access needs. Therefore, organizations must establish countermeasures to protect their systems and resources.

1.2.4 Countermeasures

Once threats and vulnerabilities are identified, countermeasures are developed to reduce risks to acceptable levels. Countermeasures often include physical systems, such as locks on doors, as well as procedural systems, such as requiring expenditures over a certain amount to be approved. The risk analysis identifies possible countermeasures and the acceptable cost of protection.

Systems contain vulnerabilities, and a number of threats exist that might exploit a system's weaknesses. Therefore developing a defensive strategy is of great importance. A defensive stance normally consists of four parts:

- *Security Policy* – Creating a comprehensive security policy is one of the most important steps in securing a system. The issues and details of the security policy are developed based on the results of the risk analysis. The policy states what items and resources need to be protected and includes plans and procedures for implementing protection strategies.

- *Prevention* – If possible, security incidents should be prevented by removing system vulnerabilities and alleviating threats. The first step to prevention is correct design, development, and installation of the system. Then the system is examined and reviewed for possible vulnerabilities, and discovered weaknesses are corrected when feasible. To alleviate accidental threats, users and operators are educated on proper usage of the system. To alleviate malicious threats, physical and logical perimeters to the system are established, and the system is monitored to deter intrusive activity.

- *Detection* – All security incidents cannot be prevented for a number of reasons. Systems contain vulnerabilities that cannot easily or quickly be found and fixed, building secure systems without any vulnerabilities is extremely difficult, and secure systems are still vulnerable to insider abuse and mistakes [Den87]. Therefore, detection of intrusive activity and other security incidents is an important line of defense. Additionally, detection of attempted attacks, even when not successful, is important for a computer system's defense. Monitoring the computer systems to detect problems in a timely fashion minimizes damage from security incidents, establishes accountability, and deters threats.

- *Response/Recovery* – Security incidents happen; thus effective procedures for responding and recovering to normal operations are needed. Response includes procedures to stop or contain damage and gather evidence on the incident. The evidence may be used during recovery to determine extent of the damage or later for criminal prosecution. After responding to the incident, the system needs to be restored and returned to normal operation. Additionally, recovery may involve determination of the exploited weaknesses and subsequent removal, if possible, of the vulnerability.

1.3 Intrusion Detection

For the reasons discussed earlier, timely detection of security problems in a computer system is an issue of significant concern. Security controls cannot be relied upon to safeguard a computer system against all threats. System vulnerabilities cannot be totally eliminated in most circumstances. As long as threats and vulnerabilities exist, detection of security incidents will play an important role in computer security.

An intrusion detection system detects security violations through automated analysis of system activity. Traditionally, security officers periodically reviewed audit logs of system activity to detect break-in attempts and other suspicious activity [HH86, VL89, JDS91]. Auditing subsystems were originally developed to produce logs of system activity for accounting purposes. Activity logs, often called audit trails, contain information such as who ran which program and how much memory, disk space, and CPU time was used. Security personnel realized that accounting packages could also be used as a security tool for detection of suspicious activity [And80, Bon81, WK85].

Security personnel found that reviews of audit trails could have a significant, positive impact on system security by uncovering suspicious activity that otherwise may have gone unnoticed [And80, JDS91]. Unfortunately, security personnel also found that the volume of audit data collected made human review of comprehensive audit data impractical, if not generally impossible [VL89, IKP95]. Additionally, human review of audit trails has the drawback of detecting security attacks after-the-fact [GT96]. In many instances, the system may be compromised and the damage done before the activity is detected by the security officer.

Much of the data recorded by an auditing system is irrelevant to intrusion detection because it pertains to legitimate activity by authorized users. The idea of using a computer to reduce audit trails to a record of security-relevant activity, i.e. activity that is suspicious and may be a indicative of a security problem, was first introduced in the early 1980's [And80]. Figure 1.1 outlines the steps in intrusion detection. An intrusion detection system (IDS) reduces audit data to a record of suspicious activity that a human can feasibly review. Later, the concept of intrusion detection was expanded to real-time monitoring of a computer system where the IDS might also react to the suspicious activity in some fashion [Lun93, MHL94].

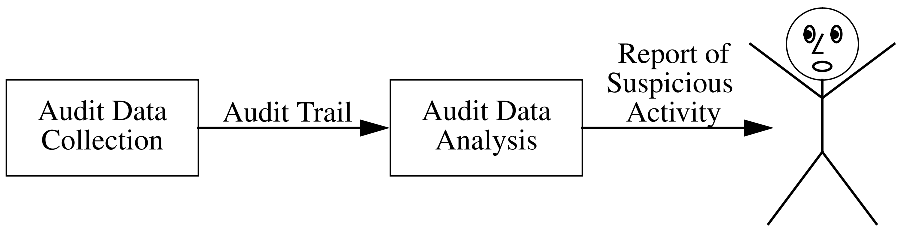

Figure 1.1 : Steps in Intrusion Detection

1.3.1 Intrusion Detection Models

In 1980, Anderson [And80] presented the idea that normal user behavior could be characterized through analysis of activity in audit trails, and computer attacks and attempted abuse could be uncovered by detecting activity that deviated significantly from the characterized "normal" behavior. After Anderson's ground-breaking paper, much research was devoted towards devising statistical methods for characterizing normal behavior and detection of abnormal activity.

By the mid-1980's, a number of prototype intrusion detection systems based on uncovering abnormal activity were under development. Haystack [Sma88] was developed for Air Force computer systems, Intrusion Detection Expert System (IDES) [LJ88] was developed at SRI International, and Discovery [MHL94] was developed at TRW. All these prototype systems were based on the hypothesis that exploitation of system vulnerabilities involves abnormal use of the system, and therefore computer attacks can be detected from abnormal system activity [Den87]. This model for intrusion detection is often referred to as the *anomaly detection model* because it tries to detect abnormal, or anomalous, activity.

In the late 1980's, intrusion detection systems began to be developed based on a second model called the *misuse detection model*. The second model is based on the hypothesis that known exploits of vulnerabilities can be described by attack signatures or patterns, and therefore computer attacks can be detected by these patterns of system misuse [MHL94]. The misuse model was developed because known intrusion techniques that involved activity within a user's characterized normal range of behavior are not detected by the anomaly detection model. Some example misuse detection systems include Los Alamos National Laboratory's Network Anomaly Detection and Intrusion Reporter (NADIR) [HJS+93], Advanced Security Audit trail Analysis on UNIX (ASAX) [HCMM92] developed at the Facultés Universitaires Notre-Dame de la Paix, and Intrusion Detection In Our Time (IDIOT) [KS94] developed at Purdue University. For more information on intrusion detection techniques, refer to [Lun88, MHL94].

1.4 Audit Trails

An audit trail is a record of computer system activity. An audit trail is generated by an auditing system that monitors system activity, and the audit trail may be comprised of user, application, and/or system-level activity. As illustrated earlier in this chapter, audit trail analysis for intrusion detection plays an important role in computer security, but audit trails also have many other significant uses in the realm of computer security [Bon81, Pic87, SM91]:

- *Maintaining Individual Accountability* – An individual's actions are tracked in an audit trail allowing users to be personally accountable for their actions. Knowing their activity is tracked leads to users being less likely to circumvent security policy, and if an incident does occur, individual accountability can be maintained.

- *Reconstructing Events* – Audit trails can be used to reconstruct the events leading up to a incident, exposing vulnerabilities in the system. The detection and removal of vulnerabilities is important to the defense of the system.
- *Assessing Damage* – Audit trails can be analyzed to determine the amount of damage that occurred with an incident. Audit data can reveal what information was disclosed or corrupted or who gained unauthorized access to information or the system.
- *Problem Monitoring* – Audit trails can be used to uncover problems and perform system health monitoring. Real-time monitoring and analysis of the status of the system allows detection of problems, such as disk failures or network outages, as they arise.
- *Deterring Computer Crime* – Belief that an effective auditing system exists and there is a significant risk of detection is a deterrent to computer crime.

The organization's security policy determines which types of activity are acceptable and which are in violation of policy. This information is used to determine what system activity is important, and thus should be monitored by the auditing system and recorded in the audit trail. Audit trails are analyzed in real-time to detect security incidents and system problems as they develop, and after-the-fact for incident recovery and damage assessment. Audit trails are often archived to allow for later analysis of crimes or incidents.

1.5 Terminology

A difficulty with intrusion detection and auditing is ambiguous terminology. Many common terms possess a number of accepted meanings. Some meanings are context-dependent, while other meanings are inconsistent. The commonly-accepted definitions are presented along with the terms used throughout the remainder of this thesis.

Audit (1) to examine a system for security problems and vulnerabilities [RS91]. (2) to record and analyze system activity for security problems and vulnerabilities [RS91]. (3) to analyze system activity for security problems [Bis89].

The three definitions are similar. The first definition refers to the activity of reviewing a static snapshot of the system. For this reason, the first defined audit activity is sometimes referred to as *static audit*. The second and third definitions refer to the dynamic, continuous activity of monitoring and analyzing the system state as it changes over time. The second definition includes both the action of creating the record of system activity and the action of analyzing the record of system activity, while the third definition only covers the analysis of the record.

Throughout the remainder of this thesis, *audit* is defined as recording and analyzing system activity for detection of security problems, and an *auditing system* is defined as a system that collects and analyzes audit data.

Audit Trail (1) A chronological set of records of system activity [RS91].

The terms *audit log*, *activity log*, and *system log* are often used interchangeably with audit trail in computer security literature.

We define an *audit trail* to be a record of system activity.

Intrusion (1) inappropriate use of a computer system [Sma88]. (2) penetration of a computer system by an outsider [HK88, JDS91].

The first definition refers to inappropriate use by either a legitimate user or an external penetrator. The second definition refers only to a computer abuse originating from outside the system. The conventional meaning of "intrusion" is entering by force [Gur84], implying an outside force breaking in.

For this thesis, *intrusion* is defined as an external penetration of the computer system.

Intrusion Detection (1) identifying individuals (users or automated attackers) who are using or attempting to use the computer system without authorization or who have legitimate access but are attempting to abuse their privileges [MHL94].

Though the term "intrusion" is sometimes limited to external penetrators only, the term "intrusion detection" is normally used to encompass identification of both insider threats and external penetrators.

For this thesis, *intrusion detection* will refer to identifying any attempted improper use of the system, whether by an authorized user or a system penetrator.

Misuse (1) inappropriate use of the computer system. (2) inappropriate use of the computer system by an insider [JDS91]. (3) known attacks that can be characterized [CS95].

The first definition refers to inappropriate use by either a legitimate user or an external penetrator, while the second definition refers only to abuse by a legitimate user. The third definition refers to activity that is detected by the misuse intrusion detection model (see section 1.3.1).

For this thesis, *misuse* is defined as an inappropriate use of the computer system.

Misuse Detection (1) identifying attempts to inappropriately use the computer system. (2) a model of intrusion detection that is based on identifying known patterns of misuse (see section 1.3.1).

For this thesis, *misuse detection* will refer to identifying any attempted improper or inappropriate use of the system. We use misuse detection and intrusion detection synonymously. When referring to the model of intrusion detection, the term *misuse detection model* is used.

Through the remainder of this thesis, the term misuse detection will be used instead of the term intrusion detection because intrusion usually connotes an external or outside attack.

1.6 A Model of an Auditing System

In this section, we develop a model of an auditing system that will serve as a framework during the review of previous work in the area of auditing systems in chapter 2. A well defined auditing system has a wide variety of potential uses in many areas including accounting, security, recovery, and privacy assurance [Bon81]. An auditing system involves the collection and analysis of information on computer system activity.

In [Bis89], Bishop presents a model of security monitoring that distinguishes between the action of collecting the data, which he terms *logging*, and the action of analyzing the data, which he terms *auditing*. In many computing systems, separate components perform each of these actions. For example, in security monitoring for misuse detection, the operating system may log the system activity to an audit trail, while a separate misuse detection system analyzes the data stored in the audit trail. Unfortunately, in much of the literature regarding misuse detection, the terms logging and auditing are used interchangeably.

In this work, we use the terms *audit data collection* and *audit data analysis* to refer to the two separate actions. We also use the term *auditing system* to encompass both the process of audit data collection and the process of audit data analysis. Figure 1.2 presents the simple model of an auditing system.

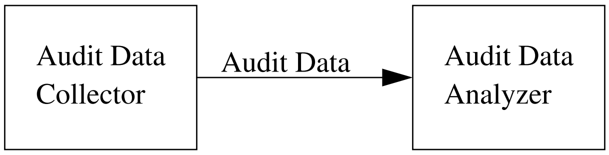

Figure 1.2 : Simple Model of an Auditing System

The structures of many early misuse detection systems closely match the simple model of an auditing system [Sma88, VL89]. Often, a file is used to transfer the audit trail between the audit data collector and the audit data analyzer. Problems arise for auditing system developers when analyzers are migrated to different audit data sources (collectors). No standard interface for audit trails is widely accepted, leading to format and content incompatibilities between different sources of audit data. An ongoing effort in the auditing research community is to develop a widely-accepted standard audit trail interface. Work towards this effort is discussed is section 2.2.

Problems arise with the simple model of an auditing system when the auditing system is extended to monitor a distributed computing system. Often, the distributed system includes multiple audit data collectors and sometimes even multiple audit data analyzers, all possibly residing on physically separate components of the system. A direct interface, such as a file, between the collectors and generators is no longer an option for communication in the distributed environment. To help alleviate the data transfer and management problems in a network environment, the concept of a distributed auditing service arose. A distributed auditing service provides mechanisms for gathering data from the audit data collector(s) and disseminating requested data to the audit data analyzer(s) throughout the distributed system. Figure 1.3 presents the model of an auditing system in a distributed environment. Distributed auditing services are discussed in section 2.3.

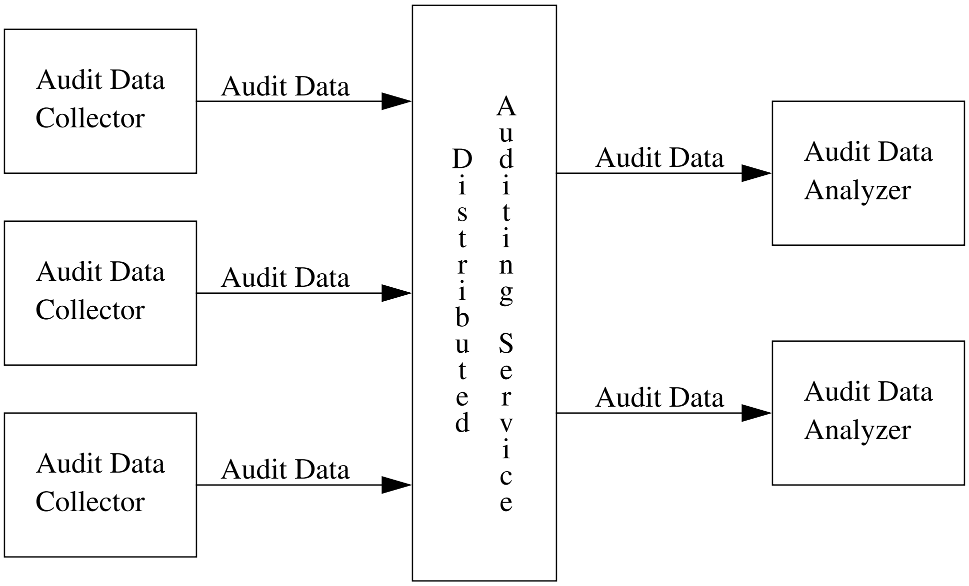

Figure 1.3 : Model of an Auditing System in a Distributed Environment

1.7 Misuse Detection Challenge of Inadequate Audit Data

A major challenge in misuse detection is that *current audit data is inadequate for misuse detection*. Audit data collected by conventional operating systems is inadequate for misuse detection in that it does not meet the needs of misuse detection. Though conventional operating system audit trails provide information that is potentially useful for uncovering attempted misuse, such audit trails omit information that is relevant to detecting misuse [Lun93, Law96, CDE+96]. Misuse detection system developers recognize that the audit data provided by conventional operating system do not meet the needs of misuse detection, but the needs of misuse detection are not fully understood and the adequacy requirements for audit trails are not well defined. A significant problem that must be solved is determining the audit data needs of misuse detection [BEF+91, SBD+91a].

To compound the problem of audit data from conventional operating systems being inadequate, different developers considered different data relevant resulting in audit trail incompatibilities [Law96]. Misuse detection systems analyze information and correlate events from multiple machines. A network computing system often consists of a variety of computer systems, each potentially having a different auditing mechanism and audit trail, further complicating the misuse detection system's task. An adequate audit trail standard that gains wide acceptance is needed to alleviate audit trail incompatibilities across machines. A few standards have been proposed and one standard, the Department of Defense's *Trusted Computer System Evaluation Criteria* [Nat85], has been widely accepted. Unfortunately, these standards do not adequately support misuse detection needs.

This work explores the above problems. A study comparing the audit data usage of a number of host-based misuse detection systems with the audit data collection capabilities of a number of conventional operating systems is presented. A number of misuse detection systems that analyze host-based audit trails are examined, and the audit data used by each system are outlined. Audit facilities of a number of conventional operating systems are also examined, and the audit data collected by each are outlined. We then compare the audit data requirements of the host-based misuse detection systems with the capabilities of the conventional operating systems. We identify the audit data that must be provided by an operating system to meet the needs of the reviewed misuse detection systems, and we examine how well the reviewed operating systems are meeting these audit data requirements. The results of this study are an important step toward the development of an audit trail with wide acceptance that adequately supports misuse detection needs. The results aid in determination of what data content should be provided by operating system audit trails to adequately support misuse detection needs.

1.8 Thesis Organization

The first chapter of this thesis provided background on computer crime, computer security, misuse detection, and audit trails. Chapter 2 discusses previous work in auditing system research. Chapter 3 examines existing misuse detection systems and the operating system generated audit data used by each. Chapter 4 examines auditing facilities of conventional operating systems and the data collected by each. Chapter 5 compares the audit data requirements of the host-based misuse detection systems with the audit data collection capabilities of the conventional operating systems. The final chapter of this thesis presents our conclusions and future work.

2. RELATED WORK

In this chapter we outline previous work performed in the area of auditing systems. In section 1.6, we developed a model of an auditing system to provide a framework for the research reviewed in this chapter. In section 2.1 prototype operating systems that incorporate extended audit data generation capabilities are examined. Next, work towards the development of a standard audit trail interface for content and format is reviewed in section 2.2. Distributed auditing services, a relatively new area of research, are presented in section 2.3. Finally, in section 2.4, unresolved auditing issues and challenges to misuse detection are discussed.

The requirements imposed on audit data by a number of audit data analysis systems for misuse detection are examined in chapter 3, while the current state of audit data collection facilities in conventional operating systems is examined in chapter 4. In the next section, the audit generation facilities of a number of prototype "secure" computing systems is reviewed.

2.1 Operating Systems with Extended Audit Generation Capabilities

A number of "secure" operating systems have been developed with enhanced security features not found in conventional systems. Many of these systems include extended audit generation capabilities. In this section, we review the goals and features of the audit generation mechanisms of a few of these secure operating systems.

Each of the following systems does provide a possible standard for an audit trail interface but, unfortunately, each of the audit trails is closely related to a particular operating system and does not easily extend to support other operating systems. The projects were not trying to develop a general-purpose audit trail interface, and their development experiences provide insight and pitfalls to avoid in auditing system development. Ideally, the successful ideas developed during the research and development of these secure operating systems will eventually transfer to conventional systems.

2.1.1 The Compartmented Mode Workstation

The Compartmented Mode Workstation (CMW) [Pic87] is a prototype secure system built by The MITRE Corporation as an extension of the UNIX 4.2BSD operating system. The two main objectives of the CMW auditing subsystem are to allow the reconstruction of events leading up to a security violation and to detect and respond to attempts to violate security. The main requirements of the CMW auditing subsystem include:

- Collect audit data from processes that have been granted the privilege to generate their own audit records.
- Collect audit data on all command information entered by users.
- Collect audit data concerning access to all objects.
- Collect audit data concerning all other security relevant events.

The system developers examined a number of data collection and data storage concerns, and also explored issues surrounding the incorporation of the auditing subsystem in the overall design of the CMW system. The CMW auditing subsystem allows pre-selection of audit data before collection, and further selective reduction of the collected data using an audit reduction program.

For each audited event, data collected include user ID, process ID, parent process ID, date, time, success/failure, and the IDs and/or sensitivity labels of any involved processes or objects. The system does not always explicitly collect the data if the data can be deduced during audit reduction from other collected data. Twenty-two types of events are audited, with each type of event being tied directly to a system routine. The direct mapping from events to auditable routines leads to the system's audit trail format being dependent on the CMW operating system, and thus also on UNIX.

In the CMW, audit collection is performed at both the application level and the operating system level. The developers felt that although the audit collection performed at the operating system level is generally sufficient to meet audit data requirements, operating system generated audit data is voluminous. Application level audit collection by trusted applications was incorporated to reduce the volume of audit data generated and to make the trails easier to comprehend. If a CMW application has been granted the ability to audit, then the operating system level audit collection is disabled for that process.

2.1.2 SunOS MLS System

SunOS MLS [Sib88] is a a secure distributed operating system developed by Sun Microsystems, targeted for evaluation at the B1 level of the Trusted Computer System Evaluation Criteria [Nat85]. The SunOS MLS system is a variant of Sun's standard SunOS operating system, version 4.0, and includes extended audit data generation capabilities. The project explores how to perform useful audit data collection in distributed systems.

A SunOS MLS system is comprised of one or more physical machines connected by a local area network. A single system image is maintained by the system independent of the individual physical machines in the system. All machines share the same file system and administrative database so file names and user identities have the same meaning regardless of location on the system. The single system image is beneficial to auditing, allowing straightforward merging of audit data across machines. System administrators view and analyze the audit trail as a system entity, even though the data was generated by numerous independent machines.

In conjunction with the single system image, the SunOS MLS system maintains a unique user identity called the *audit user ID* through all activities performed by the user between login and logout. The audit user ID is inherited by all descendants of the initial login process and is maintained when a user issues the `su` command or `rlogin` command to initiate a session on another machine. The audit user ID provides accountability back to the user who initially logged into the system.

The SunOS MLS project focuses on developing secure mechanisms for collection, transmission, storage, and merger of audit trails. The SunOS MLS project does not explore the audit data content requirements imposed by distributed processing in depth, but the project does present a standard audit file format for distributed data collection. The format allows for variable-length records, but the record fields are implicitly defined and are tied to the UNIX operating system. The presented audit file format, with fixed fields, does not allow for easy extension or modification to support other systems or goals.

2.1.3 The VAX Security Kernel

The VAX security kernel [SM90] is a prototype system developed by Digital Equipment Corporation to meet the requirements of the A1 level of the Department of Defense Trusted Computer System Evaluation Criteria [Nat85]. The security kernel is implemented as a virtual machine monitor (VMM) and has extensive audit generation capabilities.

The VAX security kernel's auditing subsystem is designed to be a production quality system while meeting the requirements of an A1 class secure system. Requirements for the auditing facility include:

- Create, maintain, and protect an audit trail.
- Support an administrative role with control over the auditing facility.
- Ability to selectively generate audit data based upon a user's identity and/or an object's security level.
- Monitor and signal impending security violations.
- Respond with defensive actions against attempted security violations.

- Record any override of human-readable labeled output.
- Audit known, auditable covert channels.

Two main types of events are audited by the VMM security kernel: object references and command references. The information recorded with each event include event name and category, event status, auxiliary data specific to the event, caller routine's name, date and time, and the subject's name, type, access class, rights, and privileges. Application level auditing is not incorporated in the project.

The project examined the many constraints placed on an auditing subsystem by the goal of meeting the requirements of an A1 class system. For example, one interesting result of architecture constraints is that the event recording an object reference always precedes the event recording the command that induced the object reference. The order is unusual and increases the work of audit reduction tools, but is consistent for all commands in the system.

The audit trail created and maintained by the VAX security kernel is limited in scope. The data collected by the auditing system is specific to one operating system, namely the VAX security kernel, and the gathered information does not support merging across platforms in a distributed system. The project does examine the many constraints imposed on A1 class systems that are not issues in conventional operating systems.

2.2 A Standard Audit Trail Interface

Development of a standard audit trail interface that adequately supports security goals and that gains wide acceptance is an extremely important step in overcoming incompatibility issues facing misuse detection systems today. Proposed audit trail standards focus on defining a standard format and/or a standard content for the data contained in an audit trail. In the following sections, proposed standards in both areas are explored.

2.2.1 Audit Trail Format Standards

We begin by reviewing proposed standards for the format of audit trails. A standard format with wide acceptance would help overcome incompatibility and interoperability problems currently faced by developers of audit data analysis systems. A standard format would allow the interchange of data from multiple audit sources and facilitate collaborative analysis of data.

2.2.1.1 Bishop's Standard Audit Trail Format

Bishop [Bis95] recognized the need to correlate audit data from many different heterogeneous systems for misuse detection. A standard format simplifies audit trail analysis, allows reconciliation of trails from differing systems, and facilitates interoperability in a networked environment. Bishop states that a standard format must be both extensible and portable to meet the needs of different, heterogeneous systems and transportability across various systems and network protocols. Bishop defines a standard log *record* format that is both portable and extensible.

To meet the need of extensibility, neither the length nor number of fields is fixed in Bishop's record format. The fields are self-defining, separated with a field separator ('#') and, delimited by start and stop symbols ('S' and 'E'). To meet the need of portability, all values are ASCII strings. This representation avoids the issues of byte ordering and floating-point format. An example log record for a UNIX command in Bishop's format could look like:

```
#time=234627364#login_id=bishop#role=root#UID=384#file=/bin/su#I#
#devno=3#inode=2343#return=1#errorcode=26#host=toad\79\#E#
```

Bishop's format does not attempt to standardize the events or fields of an audit trail record, but rather focuses on the needs of information interchange. A common record format is an important step towards interoperability and correlation of audit data in distributed systems.

2.2.1.2 Normalized Audit Data Format (NADF)

The Normalized Audit Data Format (NADF) [Mou95, Mou97] was defined by the developers of the ASAX misuse detection system [HCMM92, MCZH95, Mou97] to provide a degree of operating system independence. Translation of native audit trails to the NADF file format allows the ASAX system to have a degree of target system independence by avoiding the need to tune the misuse detection system for every possible source of audit data.

A NADF audit trail is a sequential file of NADF records. During translation, the audit records of the native audit trail are abstracted into a sequence of audit data values. Each audit data value is stored in a separate NADF record consisting of three fields:

- *Identifier* – the type of the audit data value.
- *Length* – the length of the audit data value.
- *Value* – the audit data value.

Theoretically, any audit data value can be represented by a NADF record. The NADF format does not standardize the types of possible audit data values. The developers focused on developing a normalized format that allows straightforward translation of native files into a universal format that provides a degree of system independence.

2.2.1.3 svr4++ Common Audit Trail Interchange Format for UNIX

svr4++ [Sma94] is a proposed audit trail standard to fill the need for a common audit trail interchange format. The objectives of the standard include coverage of network and host events, support of application defined events, and support of multiple vendor versions of the UNIX operating system.

The basic structure of a svr4++ record is presented in table 2.1. The miscellaneous data field(s) allow for incorporation of additional information into the record that are not covered by the basic fields.

Table 2.1 : svr4++ Basic Record Structure

Field	**Description**
Time	Date and time.
Event Type	Audit event type.
Process ID	Process identifier
Outcome	Event outcome; success or failure.
User IDs	Full description of the subject's user identifiers.
Group IDs	Full description of the subject's group identifiers.
Session ID	Identifier of session to which the process belongs.
Security Level	MAC (Mandatory Access Control) information for the subject of the event.
Object Description	Information about the object(s) affected by the event. Description includes object name, type, access control information, etc.
Miscellaneous Data	Miscellaneous event-specific data.

The svr4++ standard is tailored to the UNIX operating system, leading to problems with non-UNIX-based platforms. Additionally, the standard specifies the basic information that should be recorded with each event, but not what events should be recorded. However, the standard never attempted to standardize the audit event types but rather focused on the problem of a need for an audit interchange format.

2.2.2 Audit Trail Content Standards

The need for an audit data interchange format is not the only problem facing auditing system developers. The content of the audit trails must also be standardized, along with format, to support interoperability.

2.2.2.1 DoD Trusted Computer System Evaluation Criteria

The DoD Trusted Computer System Evaluation Criteria (TCSEC) [Nat85, Nat87] is a standard created by the National Computer Security Center against which a computing system can be evaluated for security assurance. The criteria are used as an acquisition standard by DoD and also as a metric by organizations seeking a level of security assurance in their computing systems.

A computer system is said to be secure if it controls "access to information such that only properly authorized individuals, or processes operating on their behalf, will have access to read, write, create, or delete information" [Nat85, p. 3]. The criteria are divided into four divisions, A, B, C, and D, with systems meeting the criteria of the highest division (A) providing the best level of security assurance. Within divisions C and B, there are a number of subdivisions called classes. Classes C2 through A1 require the ability to audit security-relevant activities on the system.

Events that must be auditable in a class C2 system include use of identification and authentication mechanisms, introduction and deletion of objects, administrative actions, and other security relevant events. A few additional auditable events are required at the higher class levels. Information that must be recorded with each audited event include:

- date and time of the event
- user identifier
- type of event
- success or failure of the event
- origin of request for identification/authentication events
- name of object for object introduction/deletion events

The DoD criteria provide guidelines requiring audit as a measure of security assurance, and the criteria have been widely accepted. Systems from many vendors strive to be certified at various divisions and classes of the criteria. Unfortunately, the criteria lack specifics in many areas of audit. The criteria provide guidelines for types of events to audit, but are not detailed enough to be an adequate standard for audit trail content. Specific requirements are not given for audit content except at a superficial level. The requirement of recording "other security relevant events" is too vague and open to individual interpretation. This is exemplified by the fact that different systems certified as class C2 compliant have different audit trail formats and contents.

2.2.2.2 Security Criteria for Distributed Systems

In 1995, the Institute for Defense Analyses presented its Security Criteria for Distributed Systems [GCBD95]. The functional requirements of the criteria only apply to operating systems of distributed systems and do not address application security. The criteria cover many different aspects of audit including audit protection, audit data, and audit for covert channel handling.

The criteria require various types of events be auditable. The events are grouped into six categories:

- Access Control and Administrative Policy Events

- Data Confidentiality and Integrity Policy Events
- Non-Discretionary Policy Events
- Availability Policy Events
- Cryptographic Policy Events
- Default and Dependent Events

The criteria also specify data to record with each event. Basic data that must be recorded include date and time, subject attribute information, identity of host generating the audit record, event class and event identifier within the class, and event outcome (success or failure). In addition, audit records for object creation and destruction must include the name and policy attributes of the object. Authentication audit records must include the subject's authentication status and subject's system entry attributes.

The criteria briefly discuss audit requirements for monitoring, and in some cases, detecting violations of security policy. The criteria leave the chore of deciding what actions should be audited to an organization's administrators and security policy developers. The criteria also state that the information technology developers must anticipate which actions are likely to be considered important by security policies and to provide mechanisms to record these actions.

The Security Criteria for Distributed Systems attempt to solve the problem of a need for a standard for audit trail content but fall short. The criteria are much too broad on coverage of content for audit trails, and too much is left to the applications developer and security officer. The guidelines for what should be audited and why are not provided except in a few select cases.

2.3 Distributed Auditing Services

Distributed auditing services are a relatively new area of research. In [SM91], many issues are discussed that effect auditing systems in a distributed environment that are not concerns in a stand-alone system. Concerns raised by Schaen and McKenney include the need for mechanisms for forwarding audit data among the components of the distributed system, and the concern that some degree of auditing system management may need to be performed remotely. A distributed auditing service provides mechanisms to address these needs.

2.3.1 Distributed Auditing System (DAS)

In [BEF+91], a Distributed Auditing System (DAS) architecture for the collection and distribution of audit data in a distributed environment is presented. The architecture addresses the issues of transporting audit data from a collection point to an analysis point, and management of audit functions from a remote location.

The DAS design consists of four major components: a virtual audit trail, an audit agent, an audit manager, and an audit data communication service. The virtual audit trail is operating system independent and is used to transport information collected by the audit agent to the audit manager. The audit data communication service provides mechanisms for transmitting the virtual audit trail and other messages between the audit agents and the audit managers.

The possibility of using network management services as a model for a distributed auditing system is examined because of the similarities between collecting audit data and collecting network performance data. The network communication services defined by the Common Management Information Service (CMIS) along with the Common Management Information Protocol (CMIP) could be adapted to provide the necessary communication services for transporting audit data between the audit agents and the audit managers.

During the development of the framework for distributed auditing many difficulties facing auditing systems in a network environment were examined, but many issues still need consideration. For example, security assurance concerns during transmission and storage of the audit data need to be addressed, and the needs of various audit analysis tools for audit data must be explored.

2.3.2 Distributed Audit Service (XDAS)

The Open Group is developing the Distributed Audit Service (XDAS) specification [Ope97] as a standard for exchanging audit information in a distributed environment. Functional requirements for a distributed audit service include:

- *Audit Event Services* – Collect, filter, and analyze audit events generated on the local host, and generate local alarms.
- *Audit Service Management* – Support remote and local administration of the audit system. These requirements are out of scope for the XDAS specification.
- *Audit Event Management* – Support configuration and disposition of alarms, provide standard calls for modification of audit selection and filtering parameters, and modify configuration of audit collection on distributed platforms.
- *Audit Log Management* – Store records in a protected audit repository and ensure that the events recorded are a reflection of what actually transpired.
- *Audit Log Enquiry* – Provide a standard format for audit trails for use by analysis applications.

The objective of the XDAS specification is to define a set of generic audit events, a portable audit record format, and APIs for submitting events to the audit service, importing audit data from existing audit services to the XDAS audit service, configuring event pre-selection criteria of the audit service, and reading records from a XDAS audit trail. The set of generic events and corresponding default event classes defined by the XDAS specification are given in table 2.2. The audit service is intended to complement existing audit services, not to replace them.

2.4 Unresolved Auditing Issues and Misuse Detection

Many issues and concerns remain to be addressed in the area of auditing systems. For example, a widely accepted standard for the format and content of audit trails is

Table 2.2 : XDAS Audit Events and Default Event Classes

Class	**Events**
Account Management Events	create, delete, disable, enable, query, and modify account
User Session Events	create, terminate, query, and modify user session
Data and Resource Management Events	create, delete, query, and modify date item
Service and Application Management Events	install, remove, configure, query, disable, and enable service or application
Service and Application Usage Events	invoke, terminate, query, and modify service or application
Peer Association Events	create, terminate, query, and modify an association; receive data and send data via an association
Data and Resource Content Access Events	create, terminate, query, and modify an association with data item; query and modify data item contents
Exceptional Events	start and shutdown system; resource exhaustion and corruption; backup and recover datastore
Audit Service Management Events	configure audit service; audit datastore full; audit datastore corrupted

needed, along with mechanisms for managing audit data collection and distribution in a network environment.

Unresolved auditing issues are a major challenge for misuse detection. A recent study of the state of the art in misuse detection [Law96] found that inconsistency in audit data from multiple sources is a fundamental challenge to misuse detection, along with a lack of understanding of what events are relevant to misuse detection. A standard for the format and content of audit trails that is adequate for misuse detection and gains wide acceptance would resolve many current challenges facing misuse detection developers.

The remaining chapters of this work present a study that addresses some of the concerns facing misuse detection. In chapter 3, a number of misuse detection systems are examined and the operating system audit data used by each are summarized. In chapter 4, auditing facilities in conventional operating systems are examined and the audit data collected by each are presented. In chapter 5, we compare the audit data collected by the conventional operating systems with the needs of host-based misuse detection systems. The results of this study can serve as an aid in determining what data content should be provided by operating systems for supporting misuse detection needs.

3. SURVEY OF MISUSE DETECTION SYSTEMS

This chapter surveys misuse detection systems and the operating system audit data used by them to detect computer misuse. Analysis methods used by the surveyed systems include statistical approaches, expert systems, signature analysis, state transition analysis, and pattern-matching.

The misuse detection systems surveyed were selected based on their meeting a number of criteria. First, each surveyed system must employ operating system generated audit data to detect system misuse. Misuse detection systems that do not analyze operating system audit trails were not considered. The Distributed Intrusion Detection System (DIDS), discussed in section 3.1, analyzes both network and host-based activity, but only the host-based audit trail analysis is examined in this work. Second, each surveyed system must have information published describing its design and how it employs audit data to uncover system misuse. Many commercial systems, such as NetRanger [GT96] from WheelGroup Corporation, were not considered because of proprietary information constraints. Third, each surveyed system must have a methodology for uncovering system misuse. Some systems, such as ASAX [HCMM92, MCZH95, Mou97], focus on providing general mechanisms for data stream analysis rather than developing a specific methodology for detecting system misuse and thus were not selected. The five misuse detection system examined in this chapter are DIDS, examined in section 3.1, IDIOT, examined in section 3.2, NADIR/UNICORN, examined in section 3.3, NIDES, examined in section 3.4, and STAT/USTAT, examined in section 3.5.

In all the surveyed systems, the audit trail analysis can be viewed as having two stages, a preprocessing stage and an analysis stage. Figure 3.1 is a pictorial representation of the two stages of audit trail analysis for misuse detection. The preprocessor imports "raw" audit data generated by the operating system and converts the data into a standard representation expected by the analysis engine. The analysis engine imports the reformatted audit data from the preprocessor and analyzes the data to detect suspicious activity or misuse. Reports or other indicators of suspicious activity and outright misuse are then output by the analysis engine.

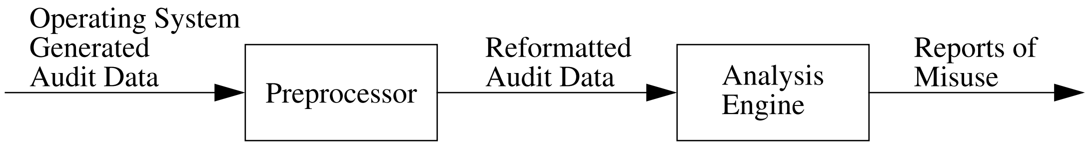

Figure 3.1 : Stages of Audit Trail Analysis in a Misuse Detection System

Our survey of misuse detection systems focuses on identifying the data used by the host-based analysis systems to detect computer misuse. For each misuse detection system, an overview is presented that discusses how the system analyzes operating system information to detect system misuse. We present the standard data format employed for misuse detection along with any additional data, not provided by the operating system audit trail, that are used by the system to detect misuse. The content of the standard formats illustrates the audit data that the system developers felt were important, and therefore should be available during analysis. The next chapter surveys the operating system sources of audit data.

3.1 The Distributed Intrusion Detection System (DIDS)

The first misuse detection system examined in this chapter is the Distributed Intrusion Detection System (DIDS) [BSD+91, SBD+91a, SBD+91b, SSTG92], a system that combines distributed monitoring and data reduction with centralized data analysis. The system monitors host machines as well as the network itself, allowing for detection of distributed attacks against a networked computer system. Aggregation and correlation of multiple data sources by DIDS allows the system to detect

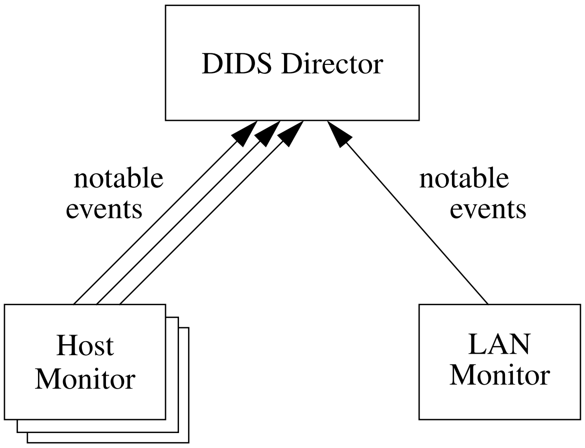

Figure 3.2 : DIDS Architecture[1]

distributed attacks whose activity recorded in any single source may not appear suspicious.

The DIDS architecture, shown in figure 3.2, includes a host monitor for each host, a LAN monitor for each LAN segment, and a central DIDS director. Each host monitor collects and analyzes audit records from the host's operating system, and notable activity is then forwarded to the director for further analysis. Similarly, the LAN monitor observes network traffic and reports notable network activity to the DIDS director for further analysis. The DIDS director employs an expert system to analyze the notable events from the host and LAN monitors and to report suspicious activity or misuse to the security officers.

The host monitor is the DIDS component of interest in this study because it is the component that analyzes operating system-generated audit data. The host monitor operates on a Sun SPARCstation running SunOS 4.1.1 or later with the Sun Basic Security Module (BSM) providing extended auditing of the system. The host monitor, whose structure is illustrated in figure 3.3, consists of an audit data preprocessor, three analysis components that act in parallel, and a communication agent for communicating with the DIDS director. The preprocessor converts BSM audit data into the standard representation expected by the analysis components. The

[1] Adapted from [SSTG92, figure 2].

notable events component filters audit data and isolates notable events to forward to the DIDS director. The *profiles* and *signature* components analyze audit data to detect local misuse and generate compilations of suspicious activity to forward to the DIDS director.

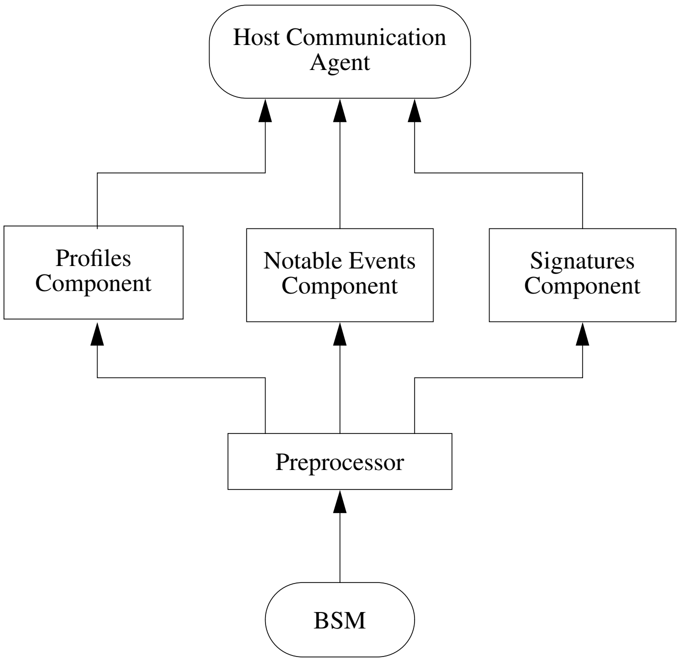

Figure 3.3 : Host Monitor Structure[2]

The profile analysis component [SSTG92] of the DIDS host monitor is actually a version of the HAYSTACK [Sma88] misuse detection system that has been modified to process data in the host audit record (HAR) format. HAYSTACK is designed to reduce voluminous audit data to short summaries of user behaviors, anomalous events, and security incidents. The profile analysis component performs two different kinds of statistical analysis. The first kind of analysis assesses characteristics of the user's behavior compared against expected characteristics of particular types of computer misuse. The second kind of analysis detects trends or tendencies in a user's behavior

[2]Adapted from [SSTG92, figure 3].

over time. The results of the statistical analysis are forwarded by the host monitor to the DIDS director for further analysis.

The signature analysis component [Sna91, SS92] of the DIDS host monitor recognizes sequences of events as previously defined attack activity. The signature analysis component analyzes the audit data, building the context surrounding each user in the system. During signature analysis, each new event is analyzed with respect to the current context. An expert system examines the current event, in the current context, to determine if one of a set of attack signatures is matched. Reports of matched signatures are forwarded to the DIDS director for further analysis.

The host monitor audit record format is discussed further in the next section. For a more detailed discussion of the overall operation of DIDS or the host monitor, refer to [SBD+91b, SSTG92].

3.1.1 The DIDS Host Monitor Audit Record Format

The host monitor preprocessor converts BSM audit records to the host audit record (HAR) format, outlined in table 3.1. Each HAR represents a single event where a subject performs an action on an object. Currently, each HAR is derived from a single BSM audit record, though the developers felt that in the future it may be useful to consider events derived from a sequence of operating system generated audit records.

The HAR format includes information provided by the BSM audit record plus two new derived data items, an *action* and a *domain*, which are intended to minimize operating system dependencies. Actions characterize activity, while domains characterize the objects of the activity. Objects include files, devices, and processes, and an object's domain is determined by its characteristics, function, or location in the system. The eleven DIDS actions are listed in table 3.2, and the ten possible domains of an object are listed in table 3.3.

Domains are ordered, and an object is assigned to the first applicable domain. Domains provide an important abstraction that allow assertions about the nature of

Table 3.1 : Host Audit Record (HAR) Format

Field
Monitor ID
Host ID
Audit UID
Real UID
Effective UID
Time
Domain
Action
Transaction
Object
Parent Process
PID
Return Value
Error Code

Table 3.2 : DIDS Actions

Action
session_start
session_end
read (a file or device)
write (a file or device)
execute (a process)
terminate (a process)
create (a file or device)
delete (a file or device)
move (rename a file or device)
change_rights
change_user_id

Table 3.3 : DIDS Domains

Domain	**Description**
tagged	objects of particular interest to intrusion detection
authentication	objects providing system access control
audit	objects relating to accounting and security auditing
network	objects relating to use of the network
system	objects relating to the operating system execution
sys_info	objects providing information about the system
user_info	objects providing information about users
utility	objects providing services to users
owned	objects relating to a user
not_owned	objects not assigned to a previous domain

a user's behavior to be made in a straightforward manner. For example, the assertion that a user is writing to an object in another user's space is easily stated and evaluated when objects are assigned to domains.

3.1.2 Audit Data Requirements Imposed by DIDS

The audit data supplied to DIDS must allow accurate derivation of profile measures for statistical analysis and detection of events and context for signature analysis. For statistical analysis, the features of a user's behavior that are profiled must be accurately monitored in the audit trail. For signature analysis, the user's context changes must be recorded in the audit record along with all occurrences of events relating to defined signatures. Additionally, information must be provided for proper determination of an object's domain.

3.2 IDIOT Pattern Matching System for Misuse Detection

The next misuse detection system examined is the generic pattern matching system for misuse detection developed by Kumar and Spafford [KS94, Kum95]. In this system, security violations are encoded in misuse signatures as interrelationships among events. Signatures are then matched against system audit trails to detect security violations. IDIOT (Intrusion Detection in our Time) [KS95, CDE+96] is an implementation of the model developed for the Solaris 2.3 operating system and the BSM audit trail.

Misuse signatures are based on the notion of events. Events are identifiable changes in some part of the system and are tagged with data. Common event tags include the time of occurrence of the event and the IDs of user(s) and object(s) associated with the event. A signature specifies the structural interrelationships among events that signify a misuse. A pattern representing a misuse signature must encode the events, context, and invariants of the signature.

Each signature may include context, which must be represented by the pattern. Accurate context specification limits false negatives and unwanted matches. The context may include preconditions that must be satisfied, along with the values of event tags. Preconditions verify that the system is in a state from which a misuse may result if the actions specified by the pattern are carried out. Patterns must also represent invariants that must hold for a misuse to result.

The generic pattern matching model represents each misuse signature as an instantiation of a Colored Petri Automaton (CPA). The interrelationships among a misuse signature's events are represented by the CPA's states and transitions. A CPA includes one or more start states and one final state, which are used to define the matching in the model. The context of event tags are represented by token colors, and preconditions are represented by guard expressions. Invariants are specified using their own CPAs, which are similar to pattern CPAs.

The audit data necessary for pattern matching is dependent on the types of patterns used for misuse detection. Kumar and Spafford developed a hierarchy for classifying misuse signatures. The classification hierarchy is:

1. *Existence* – The fact that an event occurred is sufficient to detect the misuse.

2. *Sequence* – The fact that several events happened in strict sequence is necessary to detect the misuse.

   Two special cases of this category are:

   (a) *interval* – Events happened an interval x apart.

   (b) *Duration* – Events existed or happened for not more than nor less than a certain interval of time.

3. *Regular Expression Patterns* – Signatures constructed as extended regular expressions involving events and AND as primitives.

4. *Other Patterns* – All other signatures that cannot be represented directly in earlier categories.

   Examples of patterns in this category include:

- *Patterns that Require Embedded Negation* – Patterns involving absence of a match.
- *Patterns that Involve Generalized Selection* – Patterns involving a match of any $x - y$ out of x events.

3.2.1 IDIOT Canonical Audit Trail

IDIOT's architecture separates the front-end preprocessor, which converts system-dependent audit trails to a canonical format, from the the back-end analyzer, which performs the pattern matching. The IDIOT preprocessor converts Sun BSM audit trails to the canonical audit format expected by the IDIOT analyzer.

The events supported by IDIOT's canonical audit trail are given in table 3.4. Each event includes basic information that are common to all events such as the identity of the user involved with the action. Each event also includes event-specific information that describes affected objects, such as files or processes, and provide name, identification, permission, and ownership information for the objects. The basic attributes common to all events are:

- Time the event took place.
- Real user ID.
- Effective user ID.
- Real group ID.
- Effective group ID.
- User audit ID.
- Session ID.
- Process ID.

- Return status of system call.
- Process return value.

3.2.2 Audit Data Requirements Imposed by IDIOT

As noted by Kumar and Spafford, pattern matching for misuse detection is dependent on the existence and ordering relationships of audited events. If a relevant event occurs, then its occurrence must be recorded in the audit trail along with the event's time of occurrence and length of duration. The existence, or absence in the case of negation, of an event is important for accurate matching of all patterns, and accurate timing information is necessary for ordering of events to determine the sequence of activity. If a relevant event is incorrectly audited or timing information is incorrectly recorded, then an actual misuse may not be correctly matched or a false misuse may be incorrectly matched.

The temporal ordering of events is not the only requirement of pattern specification; representation of context is also important. Context includes pre-conditions that must be met before matching the pattern, and expressions involving the tagged data values monitored with events. The audit trail must provide information such that context and pre-condition states can be accurately accessed, and the occurrence, or absence, of events of interest are accurately detected during pattern matching analysis.

3.3 The NADIR/UNICORN Misuse Detection System

The next misuse detection system examined is the Network Anomaly Detection and Intrusion Reporter (NADIR) [JDS91, HJS+93], a prototype misuse detection system developed by Los Alamos National Laboratory. NADIR employs an expert system to identify misuse scenarios in summaries of user activity. The system has been operational since 1990 at Los Alamos, monitoring a variety of network systems and services.

Table 3.4 : Events Supported by IDIOT's Canonical Audit Trail

Events
EXEC, EXECVE
LINK, SYMLINK
MKNOD
LOGIN
SU
EXIT
OPEN_R, OPEN_RC, OPEN_RT, OPEN_RTC, OPEN_RW, OPEN_RWC, OPEN_RWT, OPEN_RWTC, OPEN_W, OPEN_WC, OPEN_WT, OPEN_WTC
ACCESS
CHDIR
CLOSE
FORK, VFORK
LSTAT, STAT
UNLINK
CREAT
CHMOD
CHOWN

The target system for NADIR is the Integrated Computing Network (ICN), Los Alamos National Laboratory's main computer network. This network includes supercomputers, workstations, network service machines, local and remote terminals, and data communication interfaces, and serves over 9,000 users [HJS+93]. The ICN consists of four partitions with each partition at a defined security level. Specialized ICN service nodes enforce the partitioning and provide network services such as user authentication, access control, file access and storage, and file movement between partitions.

While many misuse detection systems monitor network traffic and/or host machines, NADIR monitors the ICN's service nodes. The three service nodes monitored by NADIR are the Network Security Controller (NSC), which provides authentication and access control, the Common File System (CFS), which maintains a partitioned file system, and the Security Assurance Machine (SAM), which authenticates all attempts to down-partition unclassified data. Each service node maintains an audit trail of its activity. These audit trails are analyzed by NADIR to detect misuse on the network. NADIR summarizes the audit data from the service nodes in weekly statistical profiles of user activity, and expert rules are then applied to the profiles to highlight suspicious activity.

In 1994, an addition to NADIR was developed called the UNICOS Real-time NADIR, or UNICORN [CJN+95], to monitor the Cray supercomputers on the ICN. UNICORN analyzes audit data collected by the UNICOS[3] operating system along with system configuration information collected by an automated security scanner. The information gathered by the security scanner allows UNICORN to identify misuse that may not be evident in the standard audit record. As with NADIR, UNICORN summarizes the audit and security scanner data in weekly statistical profiles, which are compared against expert rules expressing security violations and improper behavior.

NADIR and UNICORN maintain statistical profiles of activity for each network user and for a composite of all users. Individual profiles summarize the activity of a

[3] Cray's proprietary version of UNIX

specific user, while the composite profile summarizes the activity of the entire system. Each profile is composed of numerous fields that summarize some aspect of activity for the subject of the profile. Both NADIR and UNICORN derive the statistical profiles from system audit trails, but UNICORN also supplements the audit trails with additional information gathered by a security scanner. During analysis, profiles are evaluated using expert rules that define security violations and suspicious activity. For a more detailed discussion on how the rules are developed or applied, refer to any of [JDS91, HJS+93, CJN+95].

The audit data analyzed by NADIR are discussed in section 3.3.1, and the audit record format used by UNICORN is discussed in section 3.3.2. The additional system information collected by the security scanner for UNICORN are presented in section 3.3.3.

3.3.1 NADIR Audit Record Data

NADIR gathers audit data from three service nodes, the Network Security Controller (NFS), the Common File System (CFS), and the Security Assurance Machine (SAM). The audit records produced by the service nodes vary in format and content for two primary reasons. First, the audit records differ based on service task. NSC records contain data about logons, while CFS and NCS records contain data about file handling activities. Second, the audit records differ because the service nodes were developed separately and involve a variety of hardware configurations and operating systems. Each audit record, regardless of source, describes a single event and contains the following information:

- Unique ID for the ICN User
- Date and Time
- Accounting Charge Code
- Error Code

The remaining data in each record are service dependent and are described in table 3.5. After the audit records generated by each service node have been collected by NADIR, the data contained in each record are extracted, and the statistical profiles maintained by NADIR are updated.

Table 3.5 : ICN Service Dependent Audit Data

Service	**Service Dependent Audit Data**
NSC	Partition and ICN address of the machine from which the authentication request originated.
	Partition, classification level, and network component that the user requests to access.
CFS	Machine from which the request originated.
	Classification of the CFS session.
	Size, partition, file name, and location within the CFS directory structure of the file.
	Action requested, e.g. saving or deleting a file.
SAM	Name and CFS location of the file to be down-partitioned. where applicable.
	Partition to which the file is to be moved, where applicable.
	Action attempted, e.g. down-partition a file.

3.3.2 UNICORN Audit Record Format

The audit data generated by the UNICOS operating system is reformatted by UNICORN into a site-specific version of the svr4++ Common Audit Trail Interchange Format for UNIX [Sma94]. The developers of UNICORN decided to employ a canonical audit record format to ease expansion of the system to other UNIX operating systems. UNICORN's audit records include the basic data specified by the Common Audit Trail Interchange Format and additional site-specific data. Table 3.6 describes the UNICORN audit record format.

Table 3.6 : UNICORN Audit Record

Basic Data	
Field	**Description**
Timestamp	Date and time at which the activity occurred.
Event Type	Type of event described in this audit record.
Process ID	Current process identifier
Outcome	Event outcome. If successful, a return code indicates the type of activity. If unsuccessful, an error code indicates the type of failure.
User IDs	Full description of the subject's user identifiers.
Group IDs	Full description of the subject's group identifiers.
Session ID	Session to which the process belongs.
Security Level	Security level of the event subject.
Object Description	Information about the objects affected by the event, if any.

Site-Specific Data	
Field	**Description**
Host	Host Cray on which the attempted activity occurred.
Partition	Security partition in which the attempted activity occurred.
Event Source	Source of the activity. For example, the workstation from which the user logged on.
Compartment	Security compartment of the attempted activity.
Category	The integrity category of the attempted activity.
Activity Data	Data specific to the type of activity being reported. It describes the event itself. Each event type has its own set of possible activity data values.

3.3.3 UNICORN Security Scanner

In addition to the UNICOS system audit logs, UNICORN also collects system information with a security scanner that probes the computer system for signs of misuse and configuration vulnerabilities. The additional information gathered by the security scanner allows UNICORN to identify misuse that is not evident in the standard audit record.

The security scanner employed by UNICORN is similar in function to scanners employed by COPS [FS90] or SPI [Bar91], but it is tailored to UNICOS. The security scanner checks for system characteristics that may indicate security problems, such as:

- Files modified by a daemon.
- Minor changes in file permissions with indirect consequences.
- Files with modifications that may violate policy.
- Modifications of critical system binaries.
- Flaws in critical file formatting.
- Protection status of system directories, files, and devices.
- Incorrect anonymous FTP configuration.
- Other file access permission problems.
- Insecure daemons.
- Invalid root configuration.

After the UNICOS audit data and system information has been collected by UNICORN, the data is analyzed and aggregated in statistical profiles.

3.3.4 Audit Data Requirements Imposed by NADIR/UNICORN

Data reduction is a key part of the NADIR/UNICORN methodology. The reduction of the voluminous raw audit data into summary profiles of user activity drastically reduces the size of the database leading to easier storage, interpretation, and analysis. In addition, the profiles provide an easily understood summary of system activity that is also suitable for examination graphically. The data supplied to NADIR or UNICORN must be sufficiently detailed such that the resulting profiles accurately summarize the activity on the monitored systems.

Each event, which is described by a single audit record, is parsed independently of the rest of the event stream. Unlike systems that look for interrelationships among events, such as pattern matching or state transition analysis, NADIR and UNICORN do not require a strict ordering of events in the audit trail. NADIR and UNICORN require little context for many events, because events are often analyzed simply based on their occurrence. For example, counts are maintained for incorrect logins and incorrect file settings without regard to the context of the failed logins or flawed file settings. Even the events involving more context are normally only concerned with a few attributes characterizing the action.

3.4 NIDES Misuse Detection System

The next misuse detection system examined in this study is the Next-generation Intrusion Detection Expert System (NIDES) [Lun89, LTG+90, AFTV94, AFV95], a real-time system that observes user behavior and flags significant behavior deviations as potential misuses. NIDES employs both a statistical analysis system and a expert rule-base system for detection of misuse and suspicious behavior. NIDES is a misuse detection system that performs real-time monitoring of a distributed system. A central NIDES server analyzes audit data collected from the monitored target hosts to detect unusual and suspicious activity.

The NIDES architecture is designed so that it can be applied to any number of heterogeneous target systems. Each target system collects relevant audit data and transforms the data into NIDES' generic audit record format. The transformed audit data are then sent to the NIDES processor for analysis. NIDES has two misuse detection analysis subsystems: a statistical analysis system and a rule-based expert system. Figure 3.4 shows how the various NIDES components interact.

NIDES' statistical analysis component detects misuse by observing departures from established patterns of use. NIDES determines whether observed behavior, as reported in the audit trail, is normal with respect to past behavior, represented by a historical profile of user activity. The statistical profiles are composed of measures that characterize observed behavior. The measures determine whether recently observed activity is normal with respect to a profile of past activity. Configuration information grouping commands and objects into categories is used during statistical analysis to generalize activity of interest. As profiles are continually updated over time based on the audit data, NIDES adaptively learns expected behavior patterns. If observed activity deviates significantly from the user's historical profile, then NIDES flags the activity as a potential misuse.

NIDES also employs a rule-based expert system to detect misuse. The expert system encodes misuse scenarios and other suspicious activities in expert rules, and NIDES flags observed activity as potential misuse if the activity matches any of the encoded misuse scenarios. Rules are evaluated based on the event data in the audit trail and on system configuration information stored in a configuration file. The rules describe suspicious behavior, known system vulnerabilities, and security policy constraints.

A goal of NIDES is to be able to integrate data from numerous sources. The NIDES Beta release supports three native audit data formats: SunOS C2, Solaris BSM, and standard UNIX accounting. To facilitate integration, native format audit data are converted to a generic audit data format before being passed to the analysis subsystems. The next section discusses the NIDES audit data format. Additional configuration information is used by both the statistical and rule-based analysis components. The additional configuration information is discussed in section 3.4.2.

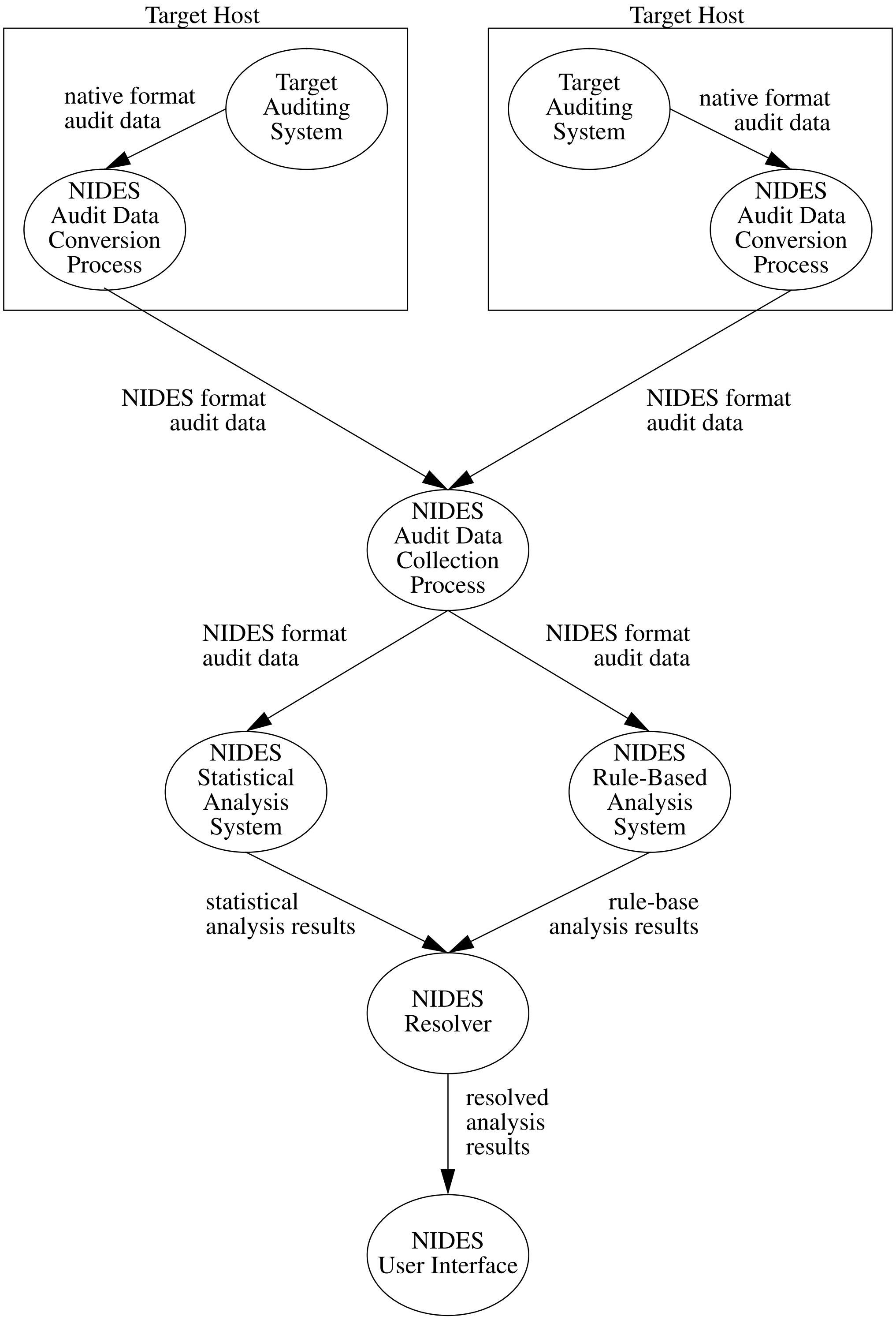

Figure 3.4 : NIDES Architecture[4]

[4]Adapted from [AFTV94, figure 2.1].

3.4.1 NIDES Audit Data Format

NIDES employs a generic audit data format to facilitate integration of data from multiple, heterogeneous sources. The data fields of a NIDES generic audit record are shown in table 3.7. Each record has a subject and an action. In the NIDES paradigm, a subject initiates actions that act on objects. Subjects are users of the system, and objects include system entities such as files and directories. A number of canonical action types are used by NIDES to aid integration of events from multiple sources. Tables 3.8 and 3.9 give the audit data actions supported by NIDES. The descriptions for the fields and actions are valid for SunOS C2, Solaris BSM, and UNIX accounting audit data, but other application data mappings may apply different meanings to some of the fields and/or actions.

3.4.2 NIDES Additional System Configuration Information

In addition to the operating system generated audit data, NIDES employs system configuration information to describe and categorize commands and objects to generalize both statistical measures and expert rules. The configuration information categorizes commands and objects into "classes" of activity for statistical analysis. The configuration information also describes and classifies many items stored in the expert system's factbase. The NIDES statistical and rule base analysis components obtain configuration information from a system configuration file, and if the system configuration changes, then the configuration file must be updated and NIDES analysis must be restarted for reconfiguration.

The class information used by the statistical analysis component are summarized in table 3.10. Classes are used by the statistical analysis component to determine categories for some statistical measures. For example, network commands and programs are grouped in the Network Commands class.

The configuration file also describes and categorizes many entities in the computing system. The NIDES expert system component obtains configuration information

Table 3.7 : NIDES Audit Data Fields

Field	**Description**
Subject	Identifier of the user.
Timestamp	Time at which the audit record was generated.
Sequence Number	Sequence number assigned to audit record.
Action	Canonical action type.
Command	Name of the command executed.
System Call	Name of the system call.
Error Number	Error value generated by the system call.
Process ID	Identifier of the process.
Return Value	Return value of the system call.
Target Host	Name of the target host.
Target Sequence Number	Sequence number assigned to audit record.
Tty	Name of the controlling tty of the process.
Arpool Timestamp	Time at which the audit record was received by arpool.
Remote Host	Name of the remote host involved with a distributed or network related operation.
Audit User Name	Real identifier of the user.
Audit User Label	Security label of the real user identifier.
User Name	Effective identifier of the user.
User label	Effective security label of the user.
Other User Name	Additional, action-specific user identifier.
Other User Label	Additional, action-specific security label.
User Time	Total CPU time spent executing non-kernel program code.
System Time	Total CPU time spent executing kernel program code.
Real Time	Total elapsed time of process.
Memory Used	Average memory usage of the process.
I/O	Number of characters transferred by process.
Read/Write	Number of blocks transferred by process.
File0	Absolute pathname of the first relevant file.
File0 Type	Type of the first file.
File0 Label	Security label of the first file.
File1	Absolute pathname of the second relevant file.
File1 Type	Type of the second file.
File1 Label	Security label of the second file.
Audit Data Source	Raw audit data type.
Argument List	Command line arguments to a command.

Table 3.8 : NIDES Audit Data Actions, Part 1

Action	Description
VOID	Unrecognized action.
DISCON	Disconnect by target host from NIDES arpool process.
ACCESS	Access a file's/directory's status information.
OPEN	Open a file/directory.
WRITE	Write or modify a file/directory.
READ	Read a file/directory.
DELETE	Delete a file.
CREATE	Create a file.
RMDIR	Remove a directory.
CHMOD	Change a file's/directory's mode.
EXEC	Execute a command.
CHOWN	Change a file's/directory's owner.
LINK	Create a symbolic link or hard link.
CHDIR	Change directories.
RENAME	Rename a file/directory.
MKDIR	Create a directory.
MOUNT	Mount a file system.
UNMOUNT	Unmount a file system.
LOGIN	Login to the system.
BAD_LOGIN	Unsuccessful login to the system.
SU	Change user identity.
BAD_SU	Unsuccessful change of user identity.
EXIT	Exit of a command.
LOGOUT	Logout from the system.
UNCAT	Not used.
RSH	Execute remote shell command.
BAD_RSH	Unsuccessful execute of remote shell command.
PASSWD	Password authentication.
RMOUNT	Remote mount.
BAD_RMOUNT	Unsuccessful remote mount.
PASSWD_AUTH	Password authentication.
BAD_PASSWD_AUTH	Unsuccessful password authentication.
KILL	Kill a process.
CORE	Dump core by process.
PTRACE	Execute `ptrace` command.
TRUNCATE	Truncate a file.
UTIMES	Execute `utimes` command.
FORK	Fork, or create, a process.

Table 3.9 : NIDES Audit Data Actions, Part 2

Action	**Description**
CHROOT	Change root directory.
MKNOD	Make a special file.
HALT	Halt the system.
REBOOT	Reboot the system.
SHUTDOWN	Shutdown the system.
BOOT	Boot the system.
SET_TIME	Change the time of the system's clock.
SET_UID	Set the user ID.
SET_GID	Set the group ID.
AUDIT_CONFIG	Change auditing configuration.
IS_PROMISCUOUS	Detect ethernet controller in promiscuous mode.
CONNECT	Connect to the target host by another host.
ACCEPT	Accept a connection by another host.
BIND	Bind a name to a socket.
SOCKET_OPTION	Action involving socket option.

Table 3.10 : NIDES Configuration Classes for Statistical Analysis

Class	**Description**
Compilers	Commands/programs that are compilers.
Editors	Commands/programs that are editors.
Mailers	Commands/programs that are associated with e-mail.
Shell Environments	Commands/programs that are shells.
Window Commands	Commands/programs associated with windows.
Network Commands	Commands/programs that are network based or remote.
Local Hosts	Hosts that are local to the monitored network.
Temporary Files	Temporary files and directories that contain temporary files.

from this file to initialize part of the expert system's factbase. The configuration information from the file used by the expert system is summarized in table 3.11.

3.4.3 Audit Data Requirements Imposed by NIDES

The operating system must provide audit data that will accurately reflect measures and allow accurate matching of rules in the rule base. The audit trail must record the occurrence of all events that may trigger rules in the expert system or change the state of any measures in the statistical profiles. Additionally, the system configuration file must accurately reflect the state of the computing system, and when the system configuration is modified, then the system configuration file must be updated.

3.5 The STAT/USTAT State Transition Analysis Tool

The final misuse detection system examined is the State Transition Analysis Tool (STAT) [Por92] for misuse detection. In state transition analysis, the actions that an attacker performs to achieve a security violation are represented by a *state transition diagram*. A state transition diagram models a penetration as a series of state changes that lead from an initial secure state to a target compromised state [Por92]. To uncover possible system misuse, a state transition analysis tool compares the state changes of the monitored computer system to the state transition diagrams of known penetrations. A UNIX-specific prototype of the tool, called USTAT, [Ilg92] was designed for SunOS 4.1.1 and the Sun C2 Basic Security Module (BSM) audit trail.

State transition analysis is intended to detect security incidents that lead to an identifiable compromised system state. Not all compromises can be identified from analysis of system attributes alone. Thus, though an incident may lead to a compromise, it may not be detectable through state transition analysis [Por92]. A security incident is representable by a state transition diagram if the incident produces a visible change in system attributes, and the compromised state is recognizable without knowledge external to the system. Types of security incidents that are representable

Table 3.11 : NIDES Configuration Information for Expert System

Class	**Description**
DOMAIN	Internet domain names of all local domains.
HOME_DIR	Users and corresponding home directories.
KNOWN_LOGIN	Accounts that are commonly left unprotected. For example, `guest` and `anonymous`.
LOG_DIR	Directories where log files are kept.
LOGIN_CONFIG	Scripts executed upon login or shell execution. For example, `.login` and `.cshrc`.
NOEXEC	Programs normal users should not run. Programs normally only run by root.
PARANOID_PROG	Programs paranoid users may execute frequently. For example, `finger` and `ps`.
PRIVATE_DEVICE	Private devices that abuser can use to eavesdrop on or spoof another user. For example, `/dev/audio`.
PRIVATE_FILE	Files that should be accessed only by the owner. For example, `.rhosts`.
PROGLOCATION	Names of directories where system files reside.
PROGRAM	System programs that should be executed only from system directories.
RAREEXEC	Programs users don't ordinarily run.
REMOTE_FILE_NO_ACCESS	Names of files that remote users should not access.
REMOTE_FILE_NO_MODIFY	Names of files that remote users should not modify.
REMOTE_NO_EXEC	Programs that remote users should not execute.
REMOTE_NOT_OK	Users who are not authorized to log in remotely.
ROOT_OK	Users authorized to become root.
SPECIAL_FILE	Files dealing with access control.
SPECIAL_PROGRAM	Special programs that should only be executed by specified users.
SPECIAL_USER	Special users that shouldn't execute anything but a specified set of programs.
SYSTEM_SCRIPTS	Shell scripts that reside in system directories.
TMP_DIRNAME	Temporary directories for the system.
TMP_FILE	Temporary files that are OK to write into the temporary directories.
USER_TYPE	Type identifiers for special users.

by state transition analysis include unauthorized access, modification, deletion, or creation of data, and unauthorized access to user or administrative privileges.

The development of a penetration scenario involves first identifying the initial and compromised states, and then identifying the key, or signature, actions necessary to move the system from the initial state, through zero or more intermediate states, to the compromised state. Signature actions are stated in the form "a *subject* performs an *action* (on an *object*)." Each state is described by state assertions about system attributes that must hold to successfully execute the penetration. The states and actions are represented graphically as a state transition diagram. Audit data is analyzed to detect signature actions and to determine if state assertions about system attributes hold as a result of the actions. The audit trail must record every occurrence of a signature action, and also provide enough information to evaluate the state assertions.

In the following sections the audit data used by STAT and USTAT are reviewed. The STAT audit record format is presented in section 3.5.1, and the audit record format of the implemented USTAT system is presented in section 3.5.2. Then additional information gathered by USTAT for testing state assertions are discussed in section 3.5.3. For additional information on the development of state transition diagrams of penetrations, refer to any of [Por92, Ilg92, IKP95].

3.5.1 STAT Audit Record Format

In STAT, audit trails generated by the computing system are preprocessed to isolate data relevant to state transition analysis. Only data corresponding directly to the signature actions and system attributes are passed for analysis. In addition, only successfully executed actions are passed, because unsuccessful actions do not produce perceivable state changes and thus are not taken into consideration during analysis. Possible actions include object read, object write, object create, object delete, program execute, program exit, modify owner object, modify object permission, modify access privilege, and login. Audit records output by the preprocessor for STAT have six fields: *Subject ID*, *Subject Permissions*, *Action*, *Object ID*, *Object Owner*, and *Object Permissions*. Table 3.12 provides a brief description of the fields.

Table 3.12 : STAT Audit Record Format

Field	**Description**
Subject ID	Unique identifier of the subject on whose behalf the audit record was generated.
Subject Permissions	Access privileges of the subject (e.g. security level, effective UID, group membership, capabilities, etc.).
Action	Action performed by the subject (on the object).
Object ID	Unique identifier of the object whose access was recorded. If no object access occurred, then this field is null.
Object Owner	Unique identifier of the owner of the object. If no object access occurred, then this field is null.
Object Permissions	The access permissions associated with the object. If no object access occurred, then this field is null.

3.5.2 USTAT Audit Record Format

In USTAT, the audit record preprocessor filters and maps BSM audit records to the USTAT audit record format. USTAT has a slightly modified audit record format from the original STAT design, because, though STAT was designed to be implemented on any computer system, some modifications during the development of the USTAT prototype were unavoidable [Ilg92]. Fields providing additional information useful for further analysis of the penetration were added, but the essential idea of having a *subject*, *action*, and *object* is preserved. Figure 3.5 gives the USTAT audit record structure.

Of the 239 different events audited by BSM, only 28 are used by USTAT. The 28 BSM events are mapped to ten different USTAT actions. Table 3.13 lists the ten different actions of USTAT along with the corresponding BSM events. The first eight

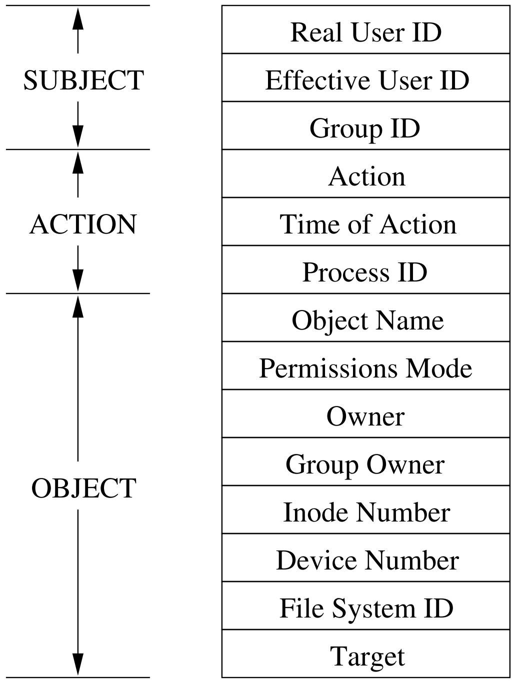

Figure 3.5 : USTAT Audit Record Structure[5]

[5]Adapted from [Ilg92, figure 4.4].

USTAT actions correspond directly to STAT actions. The last two USTAT actions, *rename* and *hardlink*, are additions to the design. *Login* was dropped from the original STAT actions during the development of USTAT.

Table 3.13 : USTAT Actions and corresponding BSM Event Types

USTAT Action	**BSM Events**
Read	open_r, open_rc, open_rtc, open_rwc, open_rwtc, open_rt, open_rw, open_rwt
Write	truncate, ftruncate, creat, open_rwc, open_rwtc, open_rw, open_rwt, open_rt, open_rtc, open_w, open_wt, open_wc, open_wtc
Create	mkdir, creat, open_rc, open_rtc, open_rwc, open_rwtc, open_wc, open_wtc, mknod
Delete	rmdir, unlink
Execute	exec, execve
Exit	exit
Modify_Owner	chown, fchown
Modify_Perm	chmod, fchmod
Rename	rename
Hardlink	link

Rename and *hardlink* pose special problems because they are actions that involve two object identifiers that represent the same physical object. Each *rename* or *hardlink* could be viewed as two actions, such as a file creation and file deletion in the case of *rename*, but the fact that the two actions referred to the same object would be obscured. Therefore, the USTAT developers decided to keep *rename* and *hardlink* as signature actions. The *login* action was dropped because the developers decided to follow the objects of the system instead of the subjects. For further details on the selection of the BSM event types and mapping to USTAT actions, refer to [Ilg92].

3.5.3 USTAT Use of Additional System Information

The USTAT analysis engine derives state actions and tests state assertions with the data from the input audit records. USTAT supports seven state assertions, listed in table 3.14. Additional information, that are not supplied by the operating system audit trails but that are needed for evaluation of state assertions, are acquired by USTAT through direct access of the computing system.

One additional source of information is the system initializer, which initializes the fact-base with *fileset* membership information. The USTAT fact-base groups files and directories that share certain characteristics into *filesets*, and the state assertion *member* asserts whether a given file is a member of a given fileset. An example fileset is all non-writable system executables. Once the fact-base is initialized, update routines keep the fact-base up-to-date with the current state of the computing system.

Another state assertion that requires additional information for evaluation is the *shell_script* assertion. The *shell_script* assertion is evaluated by opening the file and checking whether it contains "#!/bin/sh" in its first line. This dependency on the current state of the file system is problematic, especially if the analysis is done in batch mode or at a remote site, but necessary because the assertion cannot be evaluated using the audit data supplied by BSM.

3.5.4 Audit Data Requirements Imposed by STAT/USTAT

State transition analysis uses the system audit trail to monitor security-relevant state changes that occur within the system. The system audit trail must record the occurrence of all key, or signature, actions resulting in system state transitions along with the values of system attributes used in evaluation of state assertions.

3.6 Results of the Survey

All of the surveyed misuse detection systems translate operating system generated audit records into a standard format used during analysis. Each of the standard formats includes information on the action recorded by the event, the subject causing

Table 3.14 : USTAT State Assertions

State Assertion	**Description**
name($file\_var$) = $file\_name$	evaluates true if $file\_var$ matches $file\_name$.
fullname($file\_var$) = $full\_path$	evaluates true if $file\_var$ matches the pathname $full\_path$.
owner($file\_var$) = $user\_id$	evaluates true if the owner of $file\_var$ matches $user\_id$
member($file\_set$, $file\_var$)	evaluates true if $file\_var$ is a member of the file set $file\_set$.
euid = $user\_id$	evaluates true if the effective user id of the subject of the last signature action matches $user\_id$.
gid = $group\_id$	evaluates true if the group id of the subject of the audit record being processed matches $group\_id$.
permitted($perm$, $file\_var$)	evaluates true if the permission bit given as $perm$ is set in $file\_var$'s permission bits.
located(NWSD, $file\_var$)	evaluates true if $file\_var$ is located in any of the directories in the NWSD (non-writable system directories) file set.
sameuser	evaluates true if the subjects of the last two signature actions are the same.
samepid	evaluates true if the process id's of the last two signature actions are the same.
shell_script($file\_var$)	evaluates true if $file\_var$ is a shell script with the "#!/bin/sh" mechanism.

or requesting the action, and any objects effected by the action. The standard formats often include additional information such as the time the event was recorded and the audit source of the generated event.

During analysis, each misuse detection system analyzes the audit data recording the activity on the monitored host system to detect changes in the system state that may be indicative of system misuse. The statistical analysis systems, such as NADIR and NIDES, use statistical models to represent historical user activity. When a user's state differs significantly from the state represented by the historical model, then the activity is reported as a possible misuse. The pattern-matching, signature analysis, and state transition analysis systems each model the system state changes that occur with known misuse scenarios. The systems then analyze the state changes of the monitored system to detect the represented misuse scenarios.

Each of these misuse detection systems requires that the audit data contain sufficient information so that the relevant system state changes will be accurately detected. The operating system must provide enough information so that the misuse detection systems' models are accurate reflections of the state of the monitored computing system. A number of audit data insufficiencies have been noted by misuse detection system developers. Developers noted that audit data often lacks user role information, object domain information, and object content information. Application level audit data is often insufficient, along with auditing of network activity in a distributed environment. The following sections discuss these inadequacies in further detail.

3.6.1 User Role Information

Cathy Stallings [Pri96a], a developer on the NADIR misuse detection project, identified the problem of not knowing the location of a user when analyzing a user's activity. The records in the audit trail supplied to NADIR provide the user's name but not the user's location in the distributed system. Stallings noted that the policies governing a user accessing the system from a secure network connection are different from the policies governing the same user accessing the system from an unsecure dial-up link. The idea was suggested of including a *role* indicator, along with the user identifier, in the subject description to characterize how the user is accessing the computer system.

3.6.2 Tracking Users in a Distributed System

One challenge facing misuse detection in a distributed environment is tracking users and objects as they move through the system. On a single host, user identifiers facilitate user accountability on the local system, but user identifiers in conventional operating systems are not coordinated across systems. The developers of the DIDS misuse detection system try to overcome this obstacle by maintaining Network-user Identification (NID) [SSTG92] for each user to provide some degree of user accountability on a distributed system.

A network user identifier is assigned to a user upon login to the monitored system, and the identifier is preserved throughout the user's session on the distributed system. The network user identifier is maintained when the user changes his/her user identity, for example with the `su` command on a UNIX system, or when the user initiates another login session on a different host in the monitored system. With a network user identification, activity from multiple host sessions, perhaps with different account names, could be attributed to a single source.

3.6.3 Object Domain Information

Many misuse detection systems group system objects that share characteristics in sets, or domains, to increase the generality of the detection mechanisms [BSD+91, AFTV94, IKP95]. Using domains allows assertions about objects to be made in a straightforward, systematic manner, and facilitates development of general activity encodings (e.g., rules, signatures, patterns, etc.) that are applicable to all objects in the domain, rather than to one specific object.

The misuse detection system requires information to determine which domains are applicable for a given object. Some systems, such as DIDS, use the contents of the audit record to determine the domain. Other systems, such as NIDES and USTAT, rely on a system configuration initializer to initialize a database identifying members of domains. The categorization of objects into domains is often subjective and may depend on the policy for the computer system. The domain of an object may be derivable after the occurrence of an event from a source other than the audit trail, such as a configuration file. For example, whether a particular program is a "compiler" can often be derived from the information in a configuration database rather than from the audit trail. Reliance on a system configuration initializer may prove problematic, especially with off-line or remote monitoring of a computer system. The decision to derive domain information from outside sources of data rather than to deduce domain information directly from the audit trail has often been made because the information is not available in the audit trail. The question as to which domain information should be derived from static sources versus which information should be collected directly in the audit trail remains to be answered.

3.6.4 Object Content Information

Audit trails typically provide a record of system activity, but the information manipulated as a result of the activity is normally not recorded [Kum95]. For example, audit logs often record that a file is opened, and perhaps even modified, but the actual information modified is normally not recorded in the audit trail. Content information for objects manipulated by activity is required for detection of certain types of misuse. For example, the misuse detection system may need to know if a critical file's format is flawed after the files' modification to detect the development of a system vulnerability [CJN+95]. Unfortunately, data characterizing an object's information content is often absent from the audit trail.

One approach is to query the system when additional information is needed. This approach is used by a number of misuse detection systems such as IDIOT and USTAT. USTAT and IDIOT both directly access system files for evaluation of shell-script assertions. Queries for additional system information are problematic because the analysis system may not have direct access to the target system, for example, if it is being run at a remote site. Additionally, the system state may change between the time the audited event occurs and the time the system is queried for additional information.

Another approach taken by some systems is to supplement audit trails with additional information that is periodically collected from the monitored system. For example, UNICORN employs a security scanner that periodically gathers system configuration information that is used in conjunction with the operating system generated audit trail to detect system misuse. This approach is also problematic because the scanner runs periodically, and so may not provide information reflecting a system problem immediately upon the problem's emergence in the system.

3.6.5 Application Level Audit Data

Few implemented misuse detection systems employ application level audit data, possibly because such data is not commonly available with today's auditing systems, but a number of system developers [Lun90, Kum95] advocate application level auditing. Both system call level and application level auditing have advantages and disadvantages with respect to volume, complexity, and clarity of data recorded and so should be considered during the design of an auditing system.

Auditing at low-levels makes it difficult for a misuser to circumvent auditing. Anderson [And80] points out the problem of clandestine users evading detection by operating at levels of control below which audit data is collected. Unfortunately, low-level auditing gathers a voluminous, complex, and highly detailed record of system activity that may obscure the user's actions and intentions [Pic87]. Additionally, system call auditing does not supply user level data leading to problems in distinguishing which actions were requested directly by the user and which actions were initiated by a program on behalf of the user [Pic87].

Application level audit data is more compact and less voluminous, possessing user-level abstractions. It is also easier to characterize misuse activity at an application level of abstraction because developers have an intuitive "feel" for activity at this level [Lun90], and security policies, often the basis for determining if an activity is a misuse, incorporate application level abstractions [Kum95]. A disadvantage of application level auditing is that a user may obscure his/her actions through command and program aliasing resulting in difficulty ascertaining what activity is really occurring [Lun90].

3.6.6 Network Events Audit Data

The misuse detection system developers noted that auditing of network services is insufficient. For example, under Solaris BSM, *rlogin* and *ftp* events are generated, but most other services, such as YP/NIS, NFS, and SMTP, provide little or no service-level auditing [AFV95]. The lack of information is a problem for misuse detection.

The developers of NIDES noted the problem of lack of data on network services, and incorporated the data collected by TCP-wrappers to supplement operating system generated data [AFV95]. The developers of the IDIOT misuse detection system noted the difficulty with lack of information on socket-related network events [CDE+96]. The IDIOT developers suggest using a wrapper library for the socket library to generate socket-related audit data.

3.7 Summary

Many different methodologies have been developed for detecting misuse of a computer system. This chapter provided an overview of five misuse detection systems and examined the operating system data used by each. Each system transforms the operating system generated audit data into a standardized format that describes the action, subject of the action, and any involved objects. The audit data are analyzed to detect system state changes indicative of misuse. The misuse detection system developers often find the audit data supplied by the operating system to be insufficient and resort to accessing the computing system directly for additional information. Noted insufficiencies include lack of information describing the subject's role, lack of information on the object's domain and content, inadequate application level audit data, and inadequate auditing of network activity.

4. SURVEY OF CONVENTIONAL OPERATING SYSTEMS

This chapter surveys the auditing systems of a number of conventional operating systems and examines the record of system activity collected by each. The auditing systems of the surveyed operating systems record security-relevant system activity in audit trails. A security-relevant action made by a user results in the generation of an audit event that is stored in the system's audit trail. An event describes the security-relevant action and the user, the process acting on behalf of the user, and the system objects involved in the action. The event often contains additional information such as the time of the action, the user's privileges, the object's access permissions, and the name of the application or system module generating the event. For each surveyed operating system, we identify the data recorded by each auditing system. Many of the surveyed auditing systems also possess mechanisms for simple audit data analysis, but these mechanisms are not examined in this survey. At the end of the chapter, we identify some limitations of auditing systems that have been encountered by misuse detection system developers.

The operating systems were selected based on their meeting a number of criteria. First, each surveyed system must have the capability to generate audit data recording system activity. Second, each surveyed system must be "conventional" in the sense that the system is applicable to general-purpose computing and has gained acceptance in the general computing community. The secure operating systems discussed in section 2.1 were not considered because they never moved into the realm of "conventional" operating systems. Finally, each operating system must have information published describing its auditing facilities. We referred to the operating system administration and reference manuals for obtaining information for this survey.

In section 4.1, the HP-UX auditing subsystem is examined. In section 4.2, the OpenVMS auditing system is presented. In section 4.3, the BSM auditing system of the Solaris operating system is presented. In section 4.4, auditing in the the UNICOS operating system is examined. Finally, in section 4.5, auditing with the Windows NT security log is examined. In chapter 5, we identify the audit data that must be provided by an operating system to sufficiently meet the needs of the surveyed misuse detection systems. We then examine how well the surveyed operating system are meeting the audit data needs of the misuse detection systems.

4.1 HP-UX

The HP-UX trusted operating system [Hew95, Hew96], a version of the UNIX operating system developed by Hewlett Packard, provides auditing of security relevant events for analysis and detection of security breaches. The auditing system records occurrences of access by subjects to objects for detection of attempts to bypass protection mechanisms or to misuse privileges. Audit records are produced by security-relevant system calls, as well as by self-auditing applications.

Auditable events are categorized into several event types to simplify administrating the auditing subsystem. The HP-UX audit event types and associated system calls and/or self-auditing applications are given in table 4.1. Some events belong to multiple event types based on the varied operations of the system call or self-auditing application.

Each HP-UX audit record consists of a header followed by a variable-length, event-specific body. Information included in the header are:

- *Time* – Time the event completes in either success or failure.
- *Process ID* – Identifier of the process being audited.
- *Error* – Success or failure of the event.
- *Event Type* – Identifier of the type of audited activity.

Table 4.1 : HP-UX Event Types and Associated System Calls and Applications

Type	**Description**	**Associated System Calls and Applications**
admin	Log all administrative and privileged events.	stime, swapon, settimeofday, sethostid, privgrp, setevent, setaudproc, audswitch, setaudid, setdomainname, reboot, sam, audisp, audevent, audsys, audusr, chfn, chsh, passwd, pwck, init
close	Log all closings of objects.	close
create	Log all creations of objects.	creat, mknod, mkdir, semget, msgget, shmget, shmat, pipe
delete	Log all deletions of objects.	rmdir, semctl, msgctl
ipcclose	Log all IPC close events.	shutdown
ipccreat	Log all IPC create events.	socket, bind
ipcdgram	Log IPC datagram transactions.	udp user datagram
ipcopen	Log all IPC open events.	connect, accept
login	Log all logins and logouts.	login, init
modaccess	Log all access modifications other than Discretionary Access Controls.	link, unlink, chdir, setuid, setgid, chroot, setgroups, setresuid, setresgid, rename, shmctl, shmdt, newgrp
moddac	Log all modifications of object's Discretionary Access Controls.	chmod, chown, umask, fchown, fchmod, setacl, fsetacl
open	Log all openings of objects.	open, execv, ptrace, execve, truncate, ftruncate, lpsched
process	Log all operations on processes.	exit, fork, vfork, kill
removable	Log all removable media events.	smount, umount, vfsmount

For records generated by system calls, the body contains the call arguments, while for application-generated records, the body contains a high-level description of the event. To conserve space in the audit file, a PID identification record is generated when a process is initiated. The data recorded in the PID record remain the same for the lifetime of the process, and so are only recorded once per process in the audit file. The PID identification record contains the following information:

- *Process's Audit ID*
- *Real User ID*
- *Real group ID*
- *Effective user ID*
- *Effective group ID*
- *Parent's process ID*
- *Terminal ID*

A few privileged applications perform self-auditing to provide a higher-level record of system operations. The applications with self-auditing capabilities are listed in table 4.2. To reduce the amount of audit data collected, some self-auditing programs suspend auditing for the system call actions they invoke. Thus, only high-level descriptions of actions by these self-auditing applications are recorded in the audit trail.

4.2 OpenVMS VAX

The OpenVMS VAX operating system [Dig96a, Dig96b] from Digital Equipment Corporation provides an auditing system that supports monitoring of security-relevant activity. Security auditing allows administrators to monitor users' activity on the system and to reconstruct events leading up to attempts to compromise system security. A security-relevant event is any activity that involves a user's access to the system or

Table 4.2 : HP-UX Self-Auditing Applications

Application	**Description**
chfn	Change finger entry.
chsh	Change login shell.
login	The login utility.
newgrp	Change effective group.
passwd	Change password.
audevent	Select events to be audited.
audisp	Display the audit data.
audsys	Start or halt the auditing system.
audusr	Select users to be audited.
init	Change run levels, users logging off.
lpsched	Schedule line printer requests.
pwck	Password/group file checker.

to protected objects within the system. The OpenVMS auditing system can record both successful and unsuccessful security-relevant activity and allows applications to contribute security event information to the audit log.

The security-relevant events recorded by the OpenVMS auditing system are divided into a number of categories called *event classes*. The operating system audits several event classes by default, and administrators can customize which event classes are audited to meet the security needs of the local site. The event classes supported by OpenVMS are given in table 4.3.

A security audit record is generated by the auditing system in response to a security-relevant event. A security audit record consists of a header packet followed by one or more data packets, as shown in figure 4.1. The record structure allows the record to be variable-length and event-specific. The information provided in the header packet include:

- *Record Type* – Indicates the type of event that occurred.
- *Record Subtype* – Further defines the type of event that occurred.
- *Facility Code* – The code of the facility that generated the event. By default, the code is zero indicating a system-generated event.

The body is composed of a number of data packets. Each data packet has a *packet type* identifier which is used to decode the information stored in the packet.

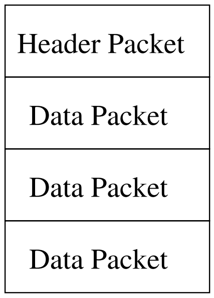

Figure 4.1 : OpenVMS Audit Record Structure

Table 4.3 : OpenVMS Event Classes

Event Class	**Description**
Access	Access requests to all objects.
ACL	Events requested by a security alarm in the ACL of an object.
Authorization	Modification of system authorization information.
Breakin	Intrusion attempts.
Connection	Logical link connections or terminations or an interprocess (IPC) call.
Create	Creation of a protected object.
Deaccess	Deaccess from a protected object.
Delete	Deletion of a protected object.
Identifier	Use of identifiers as privileges.
Install	Modifications made to the known file list through the Install utility.
Logfailure	Unsuccessful login attempts.
Login	Successful login attempts.
Logout	Logouts.
Mount	Volume mounts and dismounts.
NCP	Modification of the network configuration database using the network control program.
Privilege	Successful or unsuccessful use of privilege.
Process	Use of one or more of the process control system services.
SYSGEN	Modification of a system parameter with the System Generation utility (SYSGEN).
Time	Modification of system time.

The OpenVMS auditing system provides approximately one-hundred different types of data packets. For a complete list of the supported types of data packets, see [Dig96b]. The number and type of data packets found in a record depends on the type of event, but commonly recorded information in a OpenVMS record includes:

- *Event Time* – Date and time of the event.
- *PID* – Process identifier of the process who caused the event.
- *Process Name* – Process' name that caused the event.
- *Username* – Name of the user.
- *Terminal Name* – Name of the local terminal.
- *Image name* – Name of the image being executed when the event took place.
- *Object Name* – Object's name.
- *Object Type* – Object's type code.
- *Status* – Status of the action.

Other types of information that may be found in data packets include the command line entered by a user, user privileges, parent process attributes, and remote node information for network events. For examples of types of data appearing in different events, see [Dig96a].

4.3 Solaris

The Solaris operating system from Sun Microsystems includes a security extension called the Basic Security Module [Sun95], or BSM. The Basic Security Module provides enhanced security auditing that is designed to achieve the C2 level in the Trusted Computer System Evaluation Criteria [Nat85]. The BSM extension supports monitoring of system users through recording of security-relevant actions.

BSM auditable events fall into two categories: kernel events and user-level events. Kernel events are generated by system calls, while user-level events are generated by application software. Fourteen utility applications of the Solaris operating system perform user-level auditing. The fourteen applications are given in table 4.4. Most events are attributable to an individual user, but a few events are nonattributable because they occur at the kernel-interrupt level or before a user is identified. Over 250 different types of events are generated by various Solaris system calls and trusted applications. Each event belongs to one or more audit classes. The eighteen predefined audit classes are shown in table 4.5.

Table 4.4 : BSM Applications that Perform User-Level Auditing

allocate	at	crontab	halt
inetd	in.ftpd	login	mountd
passwd	reboot	rpc.rexd	in.rexecd
in.rshd	su		

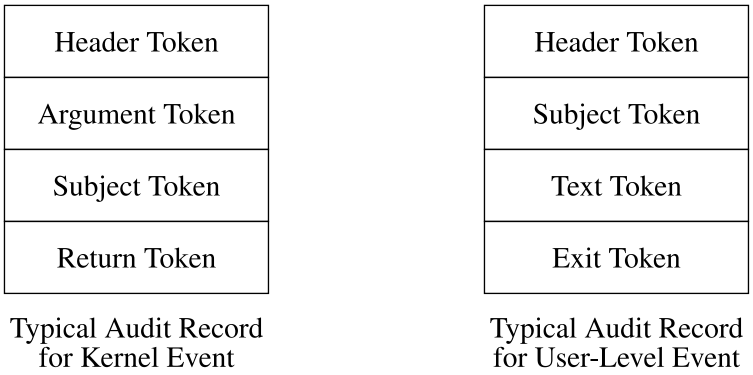

Figure 4.2 : Typical BSM Audit Records

A BSM audit record is composed of a sequence of audit tokens, with each token containing specific information about the event. There are twenty-five different audit

Table 4.5 : BSM Audit Classes

Class	**Description**
no_class	Null class for turning off event preselection.
file_read	Read of data, open for reading, etc.
file_write	Write of data, open for writing, etc.
file_attr_acc	Access of object attributes
file_attr_mod	Change of object attributes
file_creation	Creation of object
file_deletion	Deletion of object.
file_close	`close` system call events.
process	Process operation.
network	Network events.
ipc	System V IPC operations.
non_attrib	Nonattributable events.
administrative	Administrative actions.
login_logout	Login and logout events.
application	Application-defined events.
ioctl	`ioctl` system call events.

tokens, but most audit records only contain a few tokens. Figure 4.2 show typical audit records for kernel events and user-level events. Every audit record has a header token, and all attributable records have a subject token. Kernel events typically have argument and return tokens containing relevant argument information and the return status of the system call, while the user-level events typically have text and exit tokens containing application-specific data and the exit status of the program. The information given by a header token include:

- *Event ID* – Identifier of the type of audit event.
- *ID Modifier* – Modifier with descriptive information about the event type.
- *Data and Time* – Date and time the record was created.

The information given by a subject token include:

- *Audit ID* – Audit user identifier that is assigned at login and inherited by all child processes. An audit ID does not change, even when the user ID changes.
- *User ID* – Effective user ID.
- *Group ID* – Effective group ID.
- *Real User ID* – Real user ID.
- *Real Group ID* – Real group ID.
- *Process ID* – Process ID.
- *Session ID* – Session identifier assigned at login time.
- *Terminal ID* – Identifier composed of the device ID and machine ID.

BSM audit records try to be self-contained in that a record contains all relevant information about an event, and does not require other audit records for interpretation. Full path names are recorded when identifying an object, and audit IDs provide accountability back to login.

4.4 UNICOS

Auditing on a UNICOS operating system [Cra95, Cra96], Cray's proprietary version of the UNIX operating system, consists of collecting information on security-relevant events and recording the information in a security log. The security audit trail documents system activity and aids in reporting on individually accountability, security policy violations, system integrity, and sensitive information handling. Many design specifications for UNICOS security features, including security auditing, are derived from the Trusted Computer System Evaluation Criteria [Nat85].

Security auditing supports individual accountability by allowing events related to security to be traced to the responsible individuals. Audit records for security-relevant events are generated by the kernel and by trusted user-level applications. The different security log record types are summarized in table 4.6. Each audit record involves an authenticated subject and has a header followed by event-specific information. The information included in the header are:

- *Date and Time* – Date and time at which the record was entered.
- *Type* – Type of record.
- *JID* – Job identifier.
- *PID* – Process identifier.
- *Real User ID* – Real user identifier.
- *Real Group ID* – Real group identifier.
- *Effective User ID* – Effective user identifier.
- *Effective Group ID* – Effective group identifier.
- *Security Label* – Security label of the subject.
- *Object Label* – Security label of the object.

Table 4.6 : UNICOS Security Log Record Types

Record	**Description**
SLG_GO	System logging start record.
SLG_STOP	System logging stop record.
SLG_CCHG	System configuration change record.
SLG_TCHG	System time change record.
SLG_DISC_7	Discretionary access violation record.
SLG_MAND_7	Mandatory access record.
SLG_OPER	Operational access record.
SLG_LOGN	Login validation record
SLG_TAPE	Tape activity record.
SLG_EOJ	End-of-job record.
SLG_CHDIR	Change directory record.
SLG_SECSYS	Security system call record.
SLG_DAC_CHNG	`setuid` system call record.
SLG_SETUID	`su` attempt record.
SLG_SU	File transfer logging record.
SLG_FXFR	Network security violations record.
SLG_IPNET	Cray NFS request record.
SLG_NFS	Network configuration change record.
SLG_NETCF	Audit criteria selection change record.
SLG_AUDIT	NQS configuration change record
SLG_NQSCF	NQS activity record.
SLG_NQS	Trusted process activity record.
SLG_TRUST	Use of privilege record.
SLG_PRIV	Cray/REELlibrarian activity record.
SLG_CRL	Discretionary access change record.

- *Login ID* – User ID assigned at login; does not change after login.

4.5 Windows NT

The Windows NT operating system [Mic95b, Mic95a] from Microsoft Corporation includes mechanisms for recording significant events on behalf of the operating system and other applications. These mechanisms allow the system administrator to monitor events related to system security, to identify security breaches, and to determine the extent and location of any damage. Windows NT is designed for C2-level security of the Trusted Computer System Evaluation Criteria [Nat85].

Events recorded by Windows NT are related to the operating system itself or to individual applications. Windows NT provides mechanisms that allow each application to define and log its own auditable events. Windows NT groups related events into categories. The categories for the security log are given in table 4.7. Each auditing application can also define its own categories.

Table 4.7 : Windows NT Security Event Categories

Category	**Description**
Account Management	Events describing high-level changes to user accounts.
Detailed Tracking	Events providing detailed subject-tracking information.
Logon/Logoff	Events describing a logon or logoff attempt.
Object Access	Events describing access to protected objects.
Policy Changes	Events describing high-level changes to the security policy database.
Privilege Use	Events describing attempts to use privileges.
System Event	Events indicating something affecting the security of the system.

Audit events are identified by the source name of the software module or application which generated the event and by an event ID unique relative to the given source.

Many audit events also include a *handle ID*, which allows the event to be associated with future events. For example, a file open event includes an assigned handle ID which is also included in future events that involve the opened file. Information commonly recorded in an event record include:

- *Time Generated* – Time at which the record was submitted.
- *Time Written* – Time at which the record was written to the logfile.
- *Event ID* – Event identifier.
- *Event Type* – Type of event logging. Possible types are Error, Warning, Information, Success Audit, and Failure Audit.
- *Event Category* – Category of the event.
- *Source Name* – Name of the source that generated the record. For example, name of application, service, driver, or subsystem.
- *Computer Name* – Name of the computer that generated the record.
- *User SID* – Security ID of the primary user.
- *Impersonation SID* – Security ID of the client if impersonated.
- *Process ID* – Identifier of the process.

4.6 Results of the Survey

We found that each of the surveyed operating systems provided an auditing system to record security-relevant activity on the monitored computing system. The auditing mechanisms were designed to aid in detection of security violations, and many of the auditing systems' design specifications were derived from the requirements of the C2 class of the Trusted Computer System Evaluation Criteria [Nat85]. Each of the auditing systems provides a mechanism for classifying events to simplify administering of the auditing system. The information recorded by each system for a security relevant event includes the identity of the user involved in the action along with the identities of any protected system objects effected by the action. Typical information recorded in an audit record includes the user and object identifiers used by the operating system to identify the user and protected objects, the user's privilege levels, the objects' access permissions, and the identifier of the process acting on behalf of the user.

Two limitations encountered by misuse detection system developers are the lack of application level auditing in many operating systems and the difficulty of audit records that are not self-contained. In addition, we encountered the limitation of lack of sufficient documentation on the auditing facility's capabilities. The next sections discuss these limitations.

4.6.1 Application Level Audit Data

Application level audit data are not commonly available with conventional operating systems, but a number of misuse detection system developers [Lun90, Kum95] advocate application level auditing. See section 3.6.5 for a discussion of the advantages and disadvantages of each level of audit collection. All the operating systems reviewed provided mechanisms for generation of application level audit data, but few of the system's applications actually perform application level auditing. For example, both the Solaris [Sun95] and the HP-UX [Hew95] operating systems include around a dozen applications that perform high-level auditing, but many standard applications, such as editors, command interpreters, window managers, and mail tools, do not include application level auditing.

In an unusual compromise, the developers of HP-UX decided to fully trust selected applications to the extent that when the application is granted the ability to audit, the operating system level auditing is disabled for the process. Even though this results in a reduced amount of audit data, this compromise is risky as it is extremely difficult, if not generally impossible, to develop absolutely secure systems [Den87]. HP-UX is not the only system to make this compromise; the Compartmented Mode

Workstation project [Pic87], a prototype secure system reviewed in section 2.1.1, also entrusts applications to perform their own auditing in special cases.

4.6.2 Self-Contained Audit Records

One trade-off faced by auditing systems is whether to have self-contained audit records that include redundant information but do not require other records for interpretation, or to reduce audit volume by eliminating redundancy and distributing an event's information across multiple records.

Solaris BSM strives to contain all relevant information about an event in the audit event record. HP-UX, on the other hand, uses a compressed audit trail format to save space that, in some cases, results in requiring multiple records to fully interpret an event. For example, the only subject information stored in most HP-UX audit records is the process ID. To gain further information about the subject of the event, the PID Identification Record for the subject must be examined. The PID Identification Record contains information characterizing the process, such as audit ID, user ID, group ID, etc., and is only generated once per process.

Audit data is quite voluminous, but some misuse detection developers [Pri96b] feel a trade-off of space is acceptable when self-contained audit records are the result. Self-contained audit records support random-access during analysis and allow misuse detection systems to maintain less state during processing of an audit trail. An additional factor is that corruption of the audit trail does not require resynchronization of state with self-contained audit records. Corruption of a compressed audit trail could lead to loss of state information.

Another difficulty that arises when records are not self-contained is that the events triggering the storage of the supplemental information may occur before auditing is initiated. Ko [Ko96] noted that BSM only stores the name of the program executed with the exec or fork events. It is very difficult to determine the program executed for a process if the program is executed before the start of the system audit collection.

To obtain the program each process is executing, Ko resorted to inspecting kernel memory.

4.6.3 Operating System Documentation

One obstacle we encountered with every surveyed operating system was a lack of documentation on the auditing facility's capabilities. We referred to the system administration and reference manuals [Hew95, Hew96, Dig96a, Dig96b, Sun95, Cra95, Cra96, Mic95b, Mic95a] for each operating system to obtain information for this survey. All the systems provided documentation on their auditing facilities in their manuals, but the documentation omitted relevant information for many of the systems. In our opinion, Solaris provided the most complete documentation and Windows NT provided the least complete documentation. Solaris provided documentation on the content and layout of every possible audit record that may be generated by the system, while Windows NT only provided sketchy information on the types of actions that are audited and a few example audit records.

Even though we felt that Solaris provided the most extensive documentation, this documentation was not completely satisfactory. A major omission in all the operating systems' documentation was information describing what conditions result in the generation of each audit event. The Solaris BSM manual provided a description of every type of event, but the documentation omitted listing the circumstances under which the audit events would be generated. For example, the `rlogin` event is documented as being associated with the `/usr/bin/login` program but upon testing of the `login` program, we found the `rlogin` events are only generated when a user remotely logs in to the local host. If a local user remotely logs in to a remote host, then the remote login is not recorded by a `rlogin` event in the audit trail of the local host.

One area for future work would be the investigation of the systems in more depth to more fully access their capabilities. The published documentation for the operating systems was found to be insufficient for gaining a full understanding of the operating systems' capabilities. Possible means for acquiring this omitted information could include source code reviews of the systems, trial-and-error testing of running systems, and interviewing the developers of the auditing facilities of the systems.

4.7 Summary

In this chapter, we examined the auditing systems of five conventional operating systems. Each auditing system provides the ability to monitor system activity through the recording of security-relevant events. Each audit event record describes the subject performing the action and any objects involved in the action. The auditing systems were found to have different emphases for the audit trail. Some systems, such as HP-UX, strive for reduced audit volume by removing redundancy from the audit trail, while other systems, such as Solaris, strive for self-contained audit records at the expense of a greater audit data volume. All the auditing systems provide mechanisms for application-level auditing, but we found that few applications actually take advantage of the provided mechanisms. In addition, we found the manual documentation for the the auditing systems to omit relevant information about the auditing systems' capabilities.

5. MISUSE DETECTION NEEDS AND AUDIT COLLECTION CAPABILITIES

In this chapter, the audit data needs of existing host-based misuse detection systems are compared with the audit data collection capabilities of conventional operating systems. In the following sections, we identify the audit data that must be provided by operating systems to support the data usage requirements of the reviewed misuse detection systems, we discuss when state information must be recorded in the audit trail, and we reexamine the audit data collection capabilities of the reviewed operating systems.

Most of the surveyed misuse detection systems support the BSM audit trail of the Solaris operating system. Table 5.1 shows the operating system audit trails supported by each misuse detection system. The systems mainly focus on detection of misuse within UNIX systems, which may lead to unforeseen problems when other auditing sources are analyzed.

Table 5.1 : Audit Trails Supported by Each Misuse Detection System

	HP-UX	OpenVMS	Solaris	UNICOS	Windows NT	Other
DIDS			X			
IDIOT			X			
NADIR/ UNICORN				X		X
NIDES			X			X
STAT/ USTAT			X			

In section 5.1, we present the event characterization information that must be deducible from the audit trail to meet the audit data requirements of the reviewed misuse detection systems. We discuss the deduction of system state information from the audit trail in section 5.2. Finally, we reexamine the capabilities of the surveyed operating systems in section 5.3 to illustrate omissions of relevant audit data by the operating systems' audit collection facilities.

5.1 Characterization of an Event

The surveyed misuse detection systems agree on the concept that a host-based event can be characterized by an action, subject, and object(s). The event records the activity resulting from an action by a subject involving system objects. The view of an event being composed of a subject, object, and action may be widespread because this view is presented in Denning's paper introducing the Intrusion Detection Model [Den87]. Denning's Intrusion Detection Model is widely referenced by misuse detection system developers.

The surveyed operating systems also agree on the concept that a security-relevant event can be characterized by an action, subject, and object(s). The action is normally the request for or the result of a system service or high-level application activity, the subject is the user and the process acting on behalf of the user, and the objects are protected system resources. An audit record typically characterizes a subject by user name and identifiers, group membership identifiers, process identifiers, and terminal identifiers. The event is typically characterized by an action identifier. The objects are typically characterized by names, identifiers, access permissions, and locations.

An open question that remains to be answered is what constitutes a security-relevant action and when should audit events be generated for an action. Many actions on a computer system are not atomic and have the potential to generate multiple events. For example, a single write operation might generate multiple events including a "request of a write operation" event, an "initiation of a write operation" event, a number of "status of a write operation" events, and a "completion of a write operation" event. Often though, only a single "write" event is recorded in the audit trail. The types and timings of events that are recorded for an action affect the types of information that may be deduced from the audit trail in regards to the action. If only the initiations of operations are recorded, then the outcome of the operations may not be deducible from the audit trail. If an event is only generated at the completion of an operation, then suspicious operations can only be detected after-the-fact rather than at their initiation.

In the next sections, we present the event characterization information that should be deducible from an audit trail to meet the data requirements of the surveyed misuse detection systems. This set of event characterization information results from the union of all the data fields of the standard audit record representations of the surveyed misuse detection systems. Tables 5.2, 5.3, 5.4, and 5.5 summarize which misuse detection systems' standardized audit records possess which event characterization information. The subject information are discussed in section 5.1.1, the action information are discussed in section 5.1.2, the object information are discussed in section 5.1.3, and the other information, not pertaining directly to the subject, action, or object, are discussed in section 5.1.4.

5.1.1 Subject Information

A subject consists of the user and the process acting on behalf of the user. The user is often characterized by a number of user names and/or user identifiers that are acquired as the user uses the system. The computing system records the changes of user identity in the audit trail. For example, most auditing systems are capable of recording when a user acquires a different user identity. Additionally, a number of different user identities are often recorded with each event. The user identities that should be deducible for each event are:

- *Audit User Identity* – User identity acquired during login to the local host. This user identity does not change during the current session on the local host, even when the user changes his/her current identity.

Table 5.2 : Subject Attributes and Misuse Detection Systems

	DIDES	IDIOT	NADIR / UNICORN	NIDES	STAT / USTAT
Audit User Identity	X	X			
Current User Identity	X	X	X	X	X
Acting User Identity	X	X	X	X	X
User ID			X	X	X
Group IDs			X		X
Labels				X	
Security Levels			X		
Privilege					X
Session ID		X	X		
Classification Level			X		
Local Host Identity	X		X		
Local Host Security Partition			X		
Remote Host Identity			X	X	
Remote Host Classification Level			X		
Remote Host Security Partition			X		
Program Name		X			
Program Locator		X			
Program Permissions		X			
Process ID	X	X	X	X	X
Parent Process ID	X				
Charge Code			X		
Real Time				X	
User CPU Time				X	
System CPU Time				X	
Memory Usage				X	
I/O Activity				X	
TTY Number				X	
Program Type		X			X

Table 5.3 : Action Attributes and Misuse Detection Systems

	DIDES	IDIOT	NADIR / UNICORN	NIDES	STAT / USTAT
Action	X		X	X	X
Time	X	X	X		X
Status/Return Value	X	X	X	X	
Error	X	X	X	X	
Command/System Call				X	
Arguments				X	

Table 5.4 : Object Attributes and Misuse Detection Systems

	DIDES	IDIOT	NADIR / UNICORN	NIDES	STAT / USTAT
Name	X	X	X	X	X
Locator		X	X		X
Type		X		X	X
Access Permissions		X			X
Security Label				X	
Security Partition			X		
Owner		X			X
Group Owner					X
Size			X		
File System ID					X

Table 5.5 : Other Attributes and Misuse Detection Systems

	DIDES	IDIOT	NADIR / UNICORN	NIDES	STAT / USTAT
Timestamp				X	
Audit Data Source Type				X	
Sequence Number				X	

- *Current User Identity* – Currently assumed user identity on the local host. E.g., real user identity in UNIX.
- *Acting User Identity* – User identity that indicates the current privilege level. E.g., effective user identity in UNIX or impersonation identity in Windows NT.

For each user identity, the attributes that are be deducible from the audit trail are:

- *User ID* – Unique identifier used to identify the user.
- *Group IDs* – Groups to which the user identity has membership.
- *Labels* – Security labels possessed by the user identity.
- *Security Levels* – Security levels accessible to the user identity.
- *Privilege* – Access privileges, or permissions, possessed by the user identity.

Some attributes, such as labels and security levels, are not applicable to all systems, but if an attribute is applicable, then its value at the time of the recorded action should be deducible from the audit trail.

Many operating systems also provide a number of additional attributes about a particular user, such as the location of the local host in a distributed system. The network location and session attributes that should be deducible from the audit trail are:

- *Session ID* – Identifier of user's session on local host.
- *Classification Level* – Classification level of the user's session.
- *Local Host Identity* – Identity of the local host on which the action occurred.
- *Local Host Security Partition* – Security partition in which the local host resides.
- *Remote Host Identity* – Identify of the remote host involved in a distributed or network related operation.

- *Remote Host Classification Level* – Classification level of the remote host involved in a distributed or network related operation.
- *Remote Host Security Partition* – Security partition in which the remote host resides.

In addition to user identity information, a subject is characterized by a number of process attributes. The process acts on behalf of the user to perform activity on the computing system. Attributes that should be deducible from the audit trail that pertain to processes are:

- *Program Name* – Name of program file executed by the process.
- *Program Locator* – Physical locator for the program; e.g., inode number in a UNIX file system.
- *Program Permissions* – Access permission for the program.
- *Process ID* – Identifier used by the operating system to identify the process.
- *Parent Process ID* – Identifier of the parent process.
- *Charge Code* – Accounting code for charging resource utilization for the process.
- *Real Time* – Real (clock) time of process execution.
- *User CPU Time* – CPU time spent in user space by the process. E.g., time spent by a UNIX process executing non-kernel program code.
- *System CPU Time* – CPU time spent in system space by the process. E.g., time spent by a UNIX process executing kernel program code.
- *Memory Usage* – Average memory usage of the process. Often the average memory usage is recorded when a process exits, but intermediate values may also be recorded.

- *I/O Activity* – A measure of I/O activity for the process. E.g., number of characters or blocks transferred transferred during the lifetime of the process. Often the average I/O activity is recorded when a process exits, but intermediate values may also be recorded.

- *TTY Number* – TTY number associated with a UNIX process.

- *Program Type* – The type of program executed. E.g., shell script, binary file, etc.

  Note that both the IDIOT and USTAT misuse detection systems deduce type information by directly accessing the program file rather than from the system audit trail.

Some attributes, such as charge code, are not applicable to all systems, but if an attribute is applicable, then its value at the time of the recorded action should be deducible from the audit trail.

5.1.2 Action Information

An event is also characterized by the action performed by the subject. The action may be low-level, such as a system-call, or high-level, such as an application command. The attributes that should be deducible from the audit trail for the action of an event are:

- *Action* – Type of action recorded in the record.

- *Time* – Time at which the action occurred.

- *Status/Return Value* – Status or return value indicating the outcome of an action. E.g., the value returned by a system call.

- *Error* – Errors generated or resulting from the action. E.g., the error number generated by a system call.

- *Command/System Call* – Name of the command or system call that generated the audit record.

- *Arguments* – Command line arguments for commands or function arguments for function calls.

The time at which an event is generated for an action will affect the values of the action's attributes. Events that are generated at the request or initiation of an action will not have the completion or error status values of the action.

5.1.3 Object Information

The third component of an event is the object of the action. The object may be physical, such as a device, or logical, such as a process. The information that should be deducible from the audit trail will vary based on the type of object. The information that should be deducible from the audit trail for each process object is the same as the information stored for each process subject. The information that should be deducible from the audit trail for each physical object affected by the event, such as a file or device, are:

- *Name* – Name used to identify the object in the event. If the object is a file, then this field possesses its full pathname.

- *Locator* – Physical locator for the object; e.g., inode number or device number if the object resides in a UNIX file system.

- *Type* – Type of object. E.g., regular, temporary, shell-script, etc. This information is often used to help determine applicable domains for the object.

  Note that both the IDIOT and USTAT misuse detection systems deduce the type information for a file by directly accessing the file rather than from the system audit trail.

- *Access Permissions* – Type of access permitted to object.

- *Security Label* – Security label of the object.
- *Security Partition* – Security partition in which the object resides.
- *Owner* – Identity of the owner of the object.
- *Group Owner* – Group owner identity.
- *Size* – Size of object. E.g., file size in bytes.
- *File System ID* – If the object is a file, then this is the identifier of the file system in which the file resides.

In cases where the action changes an attribute of the object, such as its name or location, then the old and new attribute values should be deducible from the audit trail. Some attributes, such as security label, are not applicable to all systems, but if an attribute is applicable, then it's value at the time of the recorded action should be deducible from the audit trail.

5.1.4 Additional Information

Some additional information, that does not relate directly to the subject, action, or object of an event, should also be deducible from the audit trail. These additional information are:

- Timestamp – Time at which the audit record was generated.
- Audit Data Source Type – Type of the native audit system. E.g., Sun BSM or HP-UX.
- Sequence Number – The sequence number of the audit record.

5.2 Deduction of System State Information

As we noted in section 3.6, the misuse detection systems analyze the audit data recording the activity on the monitored computer system to detect changes in the system state that may be indicative of computer system misuse. The state of the subject, action, and objects at the time of the event occurrence must be deducible from the audit trail, but it may not be necessary for the state of all the attributes of the subject, action and objects to be recorded with each event. Many attributes' states are set before the event occurs and may have been previously recorded in the audit trail. The states of all relevant attributes to an event may not need to be recorded in every event audit record; only the identifiers necessary to identify the subject, objects, and action of the event need to be recorded along with any attributes that were effected by the action. If it is not known whether a particular piece of relevant state information was previously recorded, then the event record must contain that piece of information characterizing the current state.

As we noted in section 4.6.2, operating systems have placed differing emphasis on the importance of self-contained audit records. Self-contained audit records allow misuse detection systems to maintain less state during processing of an audit trail, but audit records with redundant state information lead to more voluminous audit trails. Audit trails with event information distributed across multiple audit records have reduced audit volume, but corruption of the audit trail can lead to resynchronization problems and loss of state information. Though audit records may not be self-contained, the audit trail should be fully self-contained. The data included in an audit trail should require no additional information from the monitored system for interpretation. Audit trails that are not fully self-contained can lead to difficulties in off-site processing. Also, the possibility exists that the state information upon which the audit data are dependent may change before that audit trail is processed.

The times at which attributes are recorded depends on a number of factors. The value of an attribute at the time the event associated with the attribute occurs must be deducible from the audit trail. If every change of state of an attribute is recorded in the audit trail, then the value of the attribute may not need to be recorded, redundantly, in the audit trail with the event record. Often redundant recording of state can be beneficial. The redundancy may be used to uncover tampering of the audit trail if the attribute states do not correspond. Redundancy can be vital if there is the possibility of loss of audit data. Another reason for redundancy may be the desire to not record the change to every system attribute, but rather to only record attribute states relevant when an event of interest occurs.

Tables 5.6, 5.7, 5.8, and 5.9 give indications of when the state of system attributes are set. If the action corresponds to the time when the state of the system attribute is set, then the attribute value must be recorded with the event. If the action does not correspond to the time when the state of the system attribute is set, then the system attribute may not need to be recorded with the event to be deducible from the audit trail.

5.3 Examination of Operating System Capabilities

In this section, we examine the audit collection capabilities of the surveyed operating systems and some omissions of relevant audit data by these systems. A misuse detection system should be able to deduce the needed subject, action, object, and other attribute state information discussed in the first sections of this chapter from the audit trail without any additional information from the monitored system. Unfortunately, none of the reviewed operating systems fully meet the audit data needs of the misuse detection systems.

Each of the surveyed operating systems' audit trails lacks some of the required state information. One area of omission for many of the operating systems in user-level information. Some of the audit trails do not support recording of user-level commands. For example, the Solaris operating does not record the commands executed by the system users. Only system calls and a handful of system utilities that are invoked by the user's commands are recorded in the audit trail. Even though some other systems, such as OpenVMS, provide mechanisms for recording user commands, many system applications do not take advantage of these mechanisms.

Another area of omission for many of the operating system audit trails is resource utilization information for processes. User CPU time, system CPU time, average

Table 5.6 : Subject Attributes and When Their States are Set

Attribute	**When State is Set**
Audit User Identity	Login to local host
Current User Identity	Last change of user name/ID
Acting User Identity	Last change of acting user name/ID
User ID	Acquisition of user identity
Group IDs	Last change of group membership for user identity
Labels	Last change of security labels for user identity
Security Levels	Last change of security levels for user identity
Privilege	Last change of privileges for user identity
Session ID	Login to local host
Classification Level	Last change of classification level for session
Local Host Identity	Login to local host
Local Host Security Partition	Last change of security partition for the local host
Remote Host Identity	Initiation of distributed action
Remote Host Classification Level	Last change of classification level for the remote host
Remote Host Security Partition	Last change of the security partition for the remote host
Program Name	Request or initiation of process
Program Locator	Last change of the the name/locator binding
Program Permissions	Last change of permission for the program
Process ID	Initiation of process
Parent Process ID	Initiation of process
Charge Code	Request or initiation of process
Real Time	Completion of process or during execution if intermediate CPU usage is recorded
User CPU Time	Completion of process or during execution if intermediate CPU usage is recorded
System CPU Time	Completion of process or during execution if intermediate CPU usage is recorded
Memory Usage	Completion of process or during execution if intermediate memory usage is recorded
I/O Activity	Completion of process or during execution if intermediate i/o usage is recorded
TTY Number	Initiation of the process
Program Type	Initiation of process

Table 5.7 : Action Attributes and When Their States are Set

Attribute	**When State is Set**
Action	Request or initiation of the action
Time	Occurrence of the action
Status/Return Value	Completion of the action or during action if intermediate status is recorded
Error	Completion of the action
Command/System Call	Request of the action
Arguments	Request of the action

Table 5.8 : Object Attributes and When Their States are Set

Attribute	**When State is Set**
Name	Request of access to the object
Locator	Access to the object
Type	Last change of the object's type
Access Permissions	Last change of the object's access permissions
Security Label	Last change of the object's security label
Security Partition	Last change of the object's security partition
Owner	Last change of the object's owner
Group Owner	Last change of the object's group owner
Size	Last modification of the object's content
File System ID	Last move of the file's location across file systems

Table 5.9 : Other Attributes and When Their States are Set

Attribute	**When State is Set**
Timestamp	Generation of the audit record
Audit Data Source Type	Initiation of the auditing process
Sequence Number	Generation of the audit record

memory usage, and I/O activity measures are not deducible from the audit trails. Many of the audit trails also lack the name/identifier binding information for system resources, and thus require access to the system for interpretation. The audit trails generated by Solaris, HP-UX, and UNICOS only contain the numeric identifier used by the operating system to identify the user. To obtain the user name associated with a particular identifier, the operating system must be queried for the current id-to-name mapping to obtain the user name.

Another common deficiency is the lack of process state information on the type of program executed. None of the operating systems collect audit information indicating if a program or file is a shell-script. USTAT and IDIOT both test to see if a program is a shell-script by directly querying the system because this state information is not provided by the audit trail. Yet another problem with object state information is ambiguity in identification. This problem was noted by the IDIOT misuse detection system developers [CDE+96] in the Solaris BSM audit trail. They noted that the name used to identify an object may be an alias, for example in the case of a link to a file, and so may differ from the actual name used by the operating system to locate the object. Problems arise when only one of the names is deducible from the audit trail.

From our survey, we determined that none of the systems fully meet the audit data needs of the misuse detection systems, and we illustrated a number of omissions of relevant audit data. We did not determine which event characterization information was deducible from each operating system's audit trail. As we discussed in section 4.6.3, we found the operating system documentation for the auditing systems to be insufficient for determination of the full extent of the systems' capabilities. Further investigation of each operating system may be useful for determining the full extent of each system's capabilities and the full extent to which each system supports the misuse detection systems event characterization needs.

5.4 Summary

In this chapter, we compared the audit data needs of existing misuse detection systems with the auditing capabilities of conventional operating systems. We identified the data that must be provided by an operating system to support the needs of the reviewed misuse detection systems, and we discussed the deduction of state information from the audit trail. We noted that although audit records may not need to be self-contained, audit trails must be self-contained. We examined the extent to which the surveyed operating systems are meeting the audit data needs of the surveyed misuse detection systems, and we noted that none of the operating systems fully meet the data needs of the misuse detection systems.

6. CONCLUSION AND FUTURE DIRECTIONS

The inadequacy of the audit data supplied by conventional operating systems is a major challenge in misuse detection. The audit data supplied by conventional operating systems lack content useful for misuse detection, and misuse detection is further impeded by the lack of a widely accepted audit trail standard that is adequate for misuse detection. This thesis provided insight into the questions of what data is needed to support a number of existing host-based misuse detection systems and how well a number of conventional operating systems are meeting these needs. We identified the audit data usage requirements for host-based analysis by a number of misuse detection systems, and we identified the audit data collection capabilities of a number of conventional operating systems. Through this study, we identified a number of specific audit data inadequacies that misuse detection system developers have encountered with existing operating systems.

We found that the misuse detection systems and operating systems agree that an event can be characterized by a subject, action and objects, but we also identified a number of specific insufficiencies in the audit data collection capabilities of conventional operating systems. We found that current auditing systems are inadequate in providing information for tracking users in a distributed system, in providing object domain and content information, and in providing information on network service activity. We found that though mechanisms for generation of application level audit data are provided by many operating systems, the application level auditing needs of the misuse detection systems are not being met because system applications are not taking advantage of the audit data generation mechanisms. In addition, we found the manuals for the operating systems to be insufficient in their coverage of the auditing facilities.

Based on what we have observed, misuse detection researchers agree that operating system audit trails are not meeting the audit data needs of misuse detection systems, and there is a lack of understanding as to what these needs are. We found that all the misuse detection system researchers we interviewed desired more information per audit record and each felt that relevant information was missing from the audit trail. We found that many developers advocated self-contained audit records which alleviate the need for a misuse detection system to maintain system state information. With the disparity in audit trail formats and contents across platforms, we found that misuse detection system developers have encountered great difficulties with migration and correlation of audit data across different systems.

6.1 Future Directions

Much work is left to be done in order to define the requirements for audit data for misuse detection. A comprehensive, widely-accepted standard that supports misuse detection needs is necessary before many problems facing today's computer security specialists can be solved. A recent study of the state of the art in misuse detection [Law96] found that inconsistency in audit data from multiple sources is a fundamental challenge to misuse detection along with a lack of understanding of what events are relevant to misuse detection.

Major problems that face misuse detection developers include understanding the audit data needs of misuse detection and defining the adequacy requirements for audit trails. This work illustrated that current audit trails omit information that is relevant to detecting misuse and exposed some of the omissions, but the significant problem of determining the audit data needs of misuse detection remains to be solved. While this work determined what data is necessary to meet the current requirements of the reviewed misuse detection systems, it does not fully lay out the audit data needs in general. The reviewed auditing systems were designed with the constraint of the limited audit data supplied by conventional operating systems. A misuse detection system designed without this constraint will have additional audit data requirements.

LIST OF REFERENCES

LIST OF REFERENCES

[AFTV94] Debra Anderson, Thane Frivold, Ann Tamaru, and Alfonso Valdes. *Next Generation Intrusion Detection Expert System (NIDES) Software Users Manual, Beta-Upgrade Release.* SRI International, Menlo Park, California, December 1994.

[AFV95] Debra Anderson, Thane Frivold, and Alfonso Valdes. Next-generation Intrusion Detection Expert System (NIDES), A Summary. Technical Report SRI-CSL-95-07, SRI International, Menlo Park, California, May 1995.

[And80] James P. Anderson. Computer Security Threat Monitoring and Surveillance. Technical report, James P. Anderson Co., Fort Washington, Pennsylvania, April 1980.

[And94] Kent E. Anderson. International Intrusions: Motives and Patterns. In *Proceedings of the 1994 Bellcore/Bell South Security Symposium*, May 1994.

[Bar91] Tony Baroletti. *Security Profile Inspector for the UNIX Operating System (SPI/UNIX).* Lawrence Livermore National Laboratory, December 1991. UCRL-MA-103440, Rev. 2.

[BEF+91] D. Banning, G. Ellingwood, C. Franklin, C. Muckenhirn, and D. Price. Auditing of Distributed Systems. In *Proceedings of the 14th National Computer Security Conference*, pages 59–68, October 1991.

[Bis89] Matt Bishop. A Model of Security Monitoring. In *Proceedings of the Fifth Annual Computer Security Applications Conference*, December 1989.

[Bis95] Matt Bishop. A Standard Audit Trail Format. In *Proceedings of the 18th National Information Systems Security Conference*, pages 136–145, October 1995.

[Bon81] David Bonyun. The Role of a Well Defined Auditing Process in the Enforcement of Privacy Policy and Data Security. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 19–25, April 1981.

[BS88] Lubomir Bic and Alan C. Shaw. *The Logical Design of Operating Systems.* Prentice Hall, second edition, 1988.

[BSD+91] James Brentano, Steven R. Snapp, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, and Stephen E. Smaha. An Architecture for a Distributed Intrusion Detection System. In *Proceedings of the 14th DOE Computer Security Group Conference*, pages 17.25–17.46, May 1991.

[CDE+96] Mark Crosbie, Bryn Dole, Todd Ellis, Ivan Krsul, and Eugene Spafford. IDIOT – Users Guide. Technical Report CSD-TR-96-050, Department of Computer Sciences, Purdue University, September 1996.

[CH96] James Cannady and Jay Harrell. A Comparative Analysis of Current Intrusion Detection Technologies. In *Proceedings of the Fourth Technology for Information Security Conference*, May 1996.

[CJN+95] Gary G. Christoph, Kathleen A. Jackson, Michael C. Neuman, Christine L. B. Siciliano, Dennis D. Simmonds, Cathy A. Stallings, and Joseph L. Thompson. UNICORN: Misuse Detection for UNICOS. In *Proceedings of the 1995 ACM/IEEE Supercomputing Conference*, December 1995.

[Cra95] Cray Research, Inc. *UNICOS Multilevel Security (MLS) Feature User's Guide*, August 1995. SG-2111 9.0.

[Cra96] Cray Research, Inc. *General UNICOS System Administration*, December 1996. SG-2301 9.2.

[CS95] Mark Crosbie and Eugene H. Spafford. Defending a Computer System using Autonomous Agents. In *Proceedings of the 18th National Information Systems Security Conference*, pages 549–558, October 1995.

[Den82] Dorothy E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[Den87] Dorothy E. Denning. An Intrusion-Detection Model. *IEEE Transactions on Software Engineering*, 13(2):222–232, February 1987.

[Dig96a] Digital Equipment Corporation, Maynard, Massachusetts. *OpenVMS Guide to System Security*, November 1996. OpenVMS VAX Version 7.1.

[Dig96b] Digital Equipment Corporation, Maynard, Massachusetts. *OpenVMS System Management Utilities Reference Manual*, November 1996. OpenVMS VAX Version 7.1.

[DR90] Cheri Dowell and Paul Ramstedt. The ComputerWatch Data Reduction Tool. In *Proceedings of the 13th National Computer Security Conference*, pages 99–108, October 1990.

[FS90] Daniel Farmer and Eugene H. Spafford. The COPS Security Checker System. In *Proceedings of the Summer 1990 USENIX Conference*, pages 163–170, June 1990.

[GCBD95] Virgil D. Gligor, Janet A. Cugini, John M. Boone, and Robert W. Dobry. *Security Criteria for Distributed Systems: Functional Requirements.* Institute for Defense Analyses, September 1995. IDA P-3159.

[GJM91] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering.* Prentice Hall, 1991.

[GS96] Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet Security.* O'Reilly & Associates, Inc., second edition, 1996.

[GT96] Bob Gleichauf and Dan Teal. *NetRanger High-Level Overview.* WheelGroup Corporation, 1996. Version 1.1.

[Gur84] David B. Guralnik, editor. *Webster's New World Dictionary of the American Language.* Simon and Schuster, second college edition, 1984.

[HA93] Stephen E. Hansen and Todd Atkins. Automated System Monitoring and Notification With Swatch. In *Proceedings of the USENIX Systems Administration (LISA VII) Conference*, pages 145–155, November 1993.

[HB95] Lawrence R. Halme and R. Kenneth Bauer. AINT Misbehaving – A Taxonomy of Anti-Intrusion Techniques. In *Proceedings of the 18th National Information Systems Security Conference*, pages 163–172, October 1995.

[HCMM92] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. ASAX: Software Architecture and Rule-Based Language for Universal Audit Trail Analysis. In *Proceedings of European Symposium on Research in Computer Security*, pages 435–450, November 1992.

[HCMM94] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathieu. Preliminary report on Advanced Security Audit Trail Analysis on uniX. Technical report, Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, September 1994.

[HDL+90] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, pages 296–303, May 1990.

[Hew95] Hewlett-Packard Company, Palo Alto, California. *HP-UX System Administration Tasks*, second edition, June 1995. B2355-90079.

[Hew96] Hewlett-Packard Company, Palo Alto, California. *HP-UX Reference Manual – Volume 4: Sections 4, 5, and 7*, third edition, July 1996. B2355-90120.

[HH86] Lawrence R. Halme and John Van Horne. Automated Analysis of Computer System Audit Trails for Security Purposes. In *Proceedings of the 9th National Computer Security Conference*, pages 71–74, September 1986.

[HJS+93] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. NADIR: An Automated System for Detecting Network Intrusion and Misuse. *Computers & Security*, 12(3):235–248, May 1993.

[HK88] Lawrence R. Halme and Brial L. Kahn. Building a Security Monitor with Adaptive User Work Profiles. In *Proceedings of the 11th National Computer Security Conference*, pages 274–283, October 1988.

[HLM+91] Richard Heady, George Luger, Arthur Maccabe, Mark Servilla, and John Sturtevant. A Prototype Implementation of a Network Level Intrusion Detection System. Technical Report CS91-11, Department of Computer Science, University of New Mexico, April 1991.

[HLMS90] Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. The Architecture of a Network Level Intrusion Detection System. Technical Report CS90-20, Department of Computer Science, University of New Mexico, August 1990.

[How97] John D. Howard. *An Analysis of Security Incidents on the Internet, 1989–1995*. PhD thesis, Department of Engineering and Public Policy, Carnegie Mellon University, April 1997.

[HWL95] James Hoagland, Christopher Wee, and Karl Levitt. Audit Log Analysis Using the Visual Audit Browser Toolkit. Technical Report CSE-95-11, Department of Computer Science, University of California, Davis, September 1995.

[III93] Walter Kiechel III. How We Will Work in the Year 2000. *Fortune*, May, 17 1993.

[IKP95] Koral Ilgun, Richard A. Kemmerer, and Phillip A. Porras. State Transition Analysis: A Rule-Based Intrusion Detection Approach. *IEEE Transactions on Software Engineering*, 21(3):181–199, March 1995.

[Ilg92] Koral Ilgun. USTAT, A Real-time Intrusion Detection System for UNIX. Master's thesis, Department of Computer Science, University of California, Santa Barbara, November 1992.

[Ilg93] Koral Ilgun. USTAT: A Real-time Intrusion Detection System for UNIX. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 16–28, May 1993.

[ISV95] David Icove, Karl Seger, and William VonStorch. *Computer Crime, A Crimefighter's Handbook*. O'Reilly & Associates, Inc., 1995.

[JDS91] Kathleen A. Jackson, David H. DuBois, and Cathy A. Stallings. An Expert System Application for Network Intrusion Detection. In *Proceedings of the 14th National Computer Security Conference*, pages 215–225, October 1991.

[JV91] Harold S. Javitz and Alfonso Valdes. The SRI IDES Statistical Anomaly Detector. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 316–326, 1991.

[JV94] Harold S. Javitz and Alfonso Valdes. The NIDES Statistical Component Description and Justification. Technical report, SRI International, Menlo Park, California, March 1994.

[Ko96] Calvin Cheuk Wang Ko. *Execution Monitoring of Security-Critical Programs in a Distributed System: A Specification-Based Approach*. PhD thesis, Department of Computer Science, University of California, Davis, 1996.

[KS94] Sandeep Kumar and Eugene H. Spafford. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference*, pages 11–21, October 1994.

[KS95] Sandeep Kumar and Eugene H. Spafford. A Software Architecture to support Misuse Intrusion Detection. Technical Report CSD-TR-95-009, Department of Computer Sciences, Purdue University, March 1995.

[Kuh86] Jeffrey D. Kuhn. Research Toward Intrusion Detection through Automated Abstraction of Audit Data. In *Proceedings of the 9th National Computer Security Conference*, pages 204–208, September 1986.

[Kum95] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Department of Computer Sciences, Purdue University, West Lafayette, Indiana, August 1995.

[Law96] Lawrence Livermore National Laboratory and Sandia National Laboratories. *National Info-Sec Technical Baseline, Intrusion Detection and Response*, December 1996. Draft.

[LJ88] Teresa F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion-Detection Expert System. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 59–66, April 1988.

[LTG+90] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Peter G. Neumann, and Caveh Jalali. IDES: A Progress Report. In *Proceedings of the 6th Annual Computer Security Applications Conference*, December 1990.

[Lun88] Teresea F. Lunt. Automated Audit Trail Analysis and Intrusion Detection: A Survey. In *Proceedings of the 11th National Computer Security Conference*, pages 65–73, October 1988.

[Lun89] Teresa F. Lunt. Real-Time Intrusion Detection. In *COMPCOM Spring '89*, pages 348–353, February/March 1989.

[Lun90] Teresa F. Lunt. IDES: An Intelligent System for Detecting Intruders. In *Proceedings of the Symposium: Computer Security, Treat and Countermeasures*, Rome, Italy, November 1990.

[Lun93] Teresa F. Lunt. A survey of intrusion detection techniques. *Computers & Security*, 12(4):405–418, June 1993.

[LV89] G. E. Liepins and H. S. Vaccaro. Anomaly Detection: Purpose and Framework. In *Proceedings of the 12th National Computer Security Conference*, pages 495–504, October 1989.

[Mar91] Victor H. Marshall. Intrusion Detection in Computers. Booz, Allen & Hamilton Inc., January 1991. Summary of the Trusted Information Systems (TIS) Report on Intrusion Detection Systems.

[MC97] Abdelaziz Mounji and Baudouin Le Charlier. Continuous Assessment of a Unix Configuration: Integrating Intrusion Detection and Configuration Analysis. In *Proceedings of the the ISOC 1997 Symposium On Network and Distributed System Security*, pages 27–35, February 1997.

[MCZH95] Abdelaziz Mounji, Baudouin Le Charlier, Denis Zampunieris, and Naji Habra. Distributed Audit Trail Analysis. In *Proceedings of the the ISOC 1995 Symposium On Network and Distributed System Security*, pages 102–112, 1995.

[MHL94] Biswanath Mukherjee, L. Todd Heberlein, and Karl N. Levitt. Network Intrusion Detection. *IEEE Network*, 8(3):26–41, May/June 1994.

[Mic95a] Microsoft Corporation. *Microsoft Win32 Programmer's Reference*, 1995.

[Mic95b] Microsoft Corporation. *Windows NT Resource Guide*, 1995.

[Mou95] Abdelaziz Mounji. *User Guide for Implementing NADF Adaptors*. Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, January 1995.

[Mou97] Abdelaziz Mounji. *Languages and Tools for Rule-Based Distributed Intrusion Detection.* PhD thesis, Institut d'Informatique, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, September 1997.

[Nat85] National Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria*, December 1985. DoD 5200.28-STD.

[Nat87] National Computer Security Center. *A Guide to Understanding Audit in Trusted Systems*, July 1987. NCSC-TG-001.

[Neu90] Peter G. Neumann. Rainbows and Arrows: How the Security Criteria Address Computer Misuse. In *Proceedings of the 13th National Computer Security Conference*, pages 414–422, October 1990.

[NP89] Peter G. Neumann and Donn B. Parker. A Summary of Computer Misuse Techniques. In *Proceedings of the 12th National Computer Security Conference*, pages 396–407, October 1989.

[Ope97] The Open Group, Berkshire, United Kingdom. *Preliminary Specification, Distributed Audit Service (XDAS) Base – Draft 8*, February 1997.

[Par94] Donn B. Parker. Demonstrating the Elements of Information Security with Threats. In *Proceedings of the 17th National Computer Security Conference*, pages 421–430, 1994.

[Pic87] J. Picciotto. The Design of An Effective Auditing Subsystem. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 13–22, April 1987.

[Por92] Phillip Andrew Porras. STAT, A State Transition Analysis Tool For Intrusion Detection. Master's thesis, Department of Computer Science, University of California, Santa Barbara, July 1992.

[Pow96] Richard Power. Current and Future Danager: A CSI Primer on Computer Crime and Information Warfare. Technical report, Computer Security Institute, 1996. second edition.

[Pow97] Richard Power. 1997 CSI/FBI Computer Crime and Security Survey. *Computer Security, Issues & Trends*, 3(2):1–13, Spring 1997. Computer Security Institute.

[Pre97] The President's Commission on Critical Infrastructure Protection. *Critical Foundations: Protecting America's Infrastructures*, October 1997. Available from `http://www.pccip.gov/`.

[Pri96a] Katherine E. Price. Interview with Cathy Stallings at Los Alamos National Laboratory, Los Alamos, New Mexico, November 1996.

[Pri96b] Katherine E. Price. Interview with Matt Bishop, Karl Levitt, Steven Smaha, Gene Spafford, and Christopher Wee at CMAD IV, Monterey, California, November 1996.

[Pro94] Paul Proctor. Audit Reduction and Misuse Detection in Heterogeneous Environments: Framework and Applications. In *Proceedings of the 10th Annual Computer Security Applications Conference*, pages 117–125, December 1994.

[PZC+96] Nicholas J. Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, and Ronald A. Olsson. A Methodology for Testing Intrusion Detection Systems. *IEEE Transactions on Software Engineering*, 22(10):719–729, October 1996.

[RS91] Deborah Russel and G.T. Gangemi Sr. *Computer Security Basics*. O'Reilly & Associates, Inc., 1991.

[SBD+91a] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, Tim Grance, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Douglass L. Mansur, Kenneth L. Pon, and Stephen E. Smaha. A System for Distributed Intrusion Detection. In *COMPCOM Spring '91*, pages 170–176, February 1991.

[SBD+91b] Steven R. Snapp, James Brentano, Gihan V. Dias, Terrance L. Goan, L. Todd Heberlein, Che-Lin Ho, Karl N. Levitt, Biswanath Mukherjee, Stephen E. Smaha, Tim Grance, Daniel M. Teal, and Doug Mansur. DIDS (Distributed Intrusion Detection System) – Motivation, Architecure, and An Early Prototype. In *Proceedings of the 14th National Computer Security Conference*, pages 167–176, October 1991.

[Sib88] W. Olin Sibert. Auditing in a Distributed System: SunOS MLS Audit Trails. In *Proceedings of the 11th National Computer Security Conference*, pages 82–90, October 1988.

[SM90] Kenneth F. Seiden and Jeffrey P. Melanson. The Auditing Facility for a VMM Security Kernel. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 262–277, May 1990.

[SM91] Samuel I. Schaen and Brian W. McKenney. Network Auditing: Issues and Recommendations. In *Proceedings of the 7th Annual Computer Security Applications Conference*, pages 66–78, December 1991.

[Sma88] Stephen E. Smaha. Haystack: An Intrusion Detection System. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44, December 1988.

[Sma94] Stephen E. Smaha. svr4++, A Common Audit Trail Interchange Format for UNIX. Technical report, Haystack Laboratories, Inc., Austin, Texas, October 1994. Version 2.2.

[Sna91] Steven Ray Snapp. Signature Analysis and Communication Issues in a Distributed Intrusion Detection System. Master's thesis, Department of Computer Science, University of California, Davis, September 1991.

[SS92] Steven R. Snapp and Stephen E. Smaha. Signature Analysis Model Definition and Formalism. In *Proceedings of the 4th Workshop on Computer Security Incident Handling and Response*, August 1992.

[SSHW88] Michael M. Sebring, Eric Shellhouse, Mary E. Hanna, and R. Alan Whitehurst. Expert Systems in Intrusion Detection: A Case Study. In *Proceedings of the 11th National Computer Security Conference*, pages 74–81, October 1988.

[SSTG92] Steven R. Snapp, Stepphen E. Smaha, Daniel M. Teal, and Tim Grance. The DIDS (Distributed Intrusion Detection) Prototype. In *Proceedings of the Summer 1992 USENIX Conference*, pages 227–233, June 1992.

[Sto89] Cliff Stoll. *The Cuckoo's Egg*. Pocket Books, 1989.

[Sun95] Sun Microsystems, Inc., Mountain View, California. *SunSHIELD Basic Security Module Guide*, November 1995.

[Sun96] Aurobindo Sundaram. An Introduction to Intrusion Detection. *Crossroads: The ACM Student Magazine*, 2(4), April 1996.

[SW95] David G. Simmons and Ronald Wilkins. NERD Network Event Recording Device: An Automated System for Network Anomaly Detection and Notification. In *Proceedings of the the ISOC 1995 Symposium On Network and Distributed System Security*, pages 87–93, February 1995.

[Tal92] Anders Tallberg. The Property of Audit Trail. Technical Report C:252, Swedish School of Economics and Business Administration, 1992.

[Vio96] Bob Violino. The Security Facade. *Information Week*, October 26 1996.

[VL89] H. S. Vaccaro and G. E. Liepins. Detection of Anomalous Computer Session Activity. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 280–289, May 1989.

[Wet93] Bradford Rice Wetmore. Paradigms for the Reduction of Audit Trails. Master's thesis, Department of Computer Science, University of California, Davis, 1993.

[WFP96a] Gregory B. White, Eric A. Fisch, and Udo W. Pooch. *Computer System and Network Security*. CRC Press Inc., 1996.

[WFP96b] Gregory B. White, Eric A. Fisch, and Udo W. Pooch. Cooperating Security Managers: A Peer-Based Intrusion Detection System. *IEEE Network*, 10(1):20–23, January/February 1996.

[WK85] Patrick H. Wood and Stephen G. Kochan. *UNIX System Security*. Hayden Books, 1985.

[Zam96] Diego M. Zamboni. SAINT: A Security Analysis Integration Tool. In *Systems Administration, Networking and Security Conference*, pages 3–15, May 1996.