2 Background

This background chapter is meant to provide readers with a review of the literature deemed as essential to understanding subsequent chapters on the topic of Unix-based intrusion detection. Firstly, it deliberates on computer security in general, then, it proceeds to present Unix in the second section. The third section deals with security in Unix while the fourth section presents the subject of intrusion detection, which is mainly absorbed from [Amoroso 1999], in length. Finally, it addresses the topic of intrusion detection in Unix.

2.1 Computer Security

Computer security has become a subject of increasing prominence as more business and government applications are being automated and more critical and sensitive data is being stored in computers – everything from your credit card data to the Department of Defense's national security information.

2.1.1 What is Computer Security?

The *Encyclopedia of Software Engineering* [Kemmerer 1994] defines *computer security* as the protection of resources (including data and programs) from unauthorized disclosure, modification or destruction. Additionally, the system resources must also be protected (i.e., access to system services should not be denied). These computer security properties are referred to as confidentiality, integrity and availability. More accurately:

- *Confidentiality* ensures that sensitive information is not disclosed to unauthorized recipients.

- *Integrity* ensures that data and programs are modified or destroyed only in a specified and authorized manner.
- *Availability* ensures that the resources of the system will be usable whenever needed by an authorized user.

The degree to which these properties are required depends on the application. For example, the defense industry is primarily interested in confidentiality. The banking industry, on the other hand, is more interested in integrity while the telephone industry may require availability the most.

2.1.2 Why Computer Security?

Most existing systems have security flaws that render them susceptible to intrusions, penetrations, and other forms of abuse.

Dorothy Denning

Even with the best security mechanisms, we must expect that a determined adversary will be able to penetrate our defenses.

Teresa Lunt

The threat is real. According to the Computer Security Institute and the Federal Bureau of Investigation, 70% of organisations surveyed reported security breaches in 2000. This figure has increased from 42% reported in 1996. Most security experts feel that these numbers are under-inflated, as there are many motivations for organisations to avoid reporting incidents.

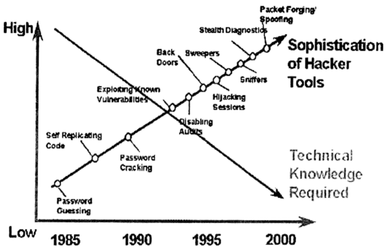


Figure 2.1: Hacker Capabilities [Cisco Systems 2000]

Moreover, as the graph (see Figure 2.1) from Cisco Systems indicate, hackers are getting smarter. In an article by Enterasys Networks [Enterasys Networks 2000], it is stated that although numerous network scanning and attacking techniques have been known for several decades, it is only recently that the tools required to perform sophisticated analysis of a target network has become publicly available. For example, the port scanners commonly available in the early 90s would simply attempt to connect to a target machine on every port to create a list of potential active ports. Modern port scanners include operating system identification, can target entire ranges of IP addresses and even transmit decoy scans to render it more difficult for the target to identify the origin of the scanner.

2.1.3 Computer Security Today

Computer security consists mainly of defensive methods used to detect and foil potential intruders. The principles of computer security hence emerge from the types of threats intruders can inflict. For example, one general security stance is that "everything that is not permitted is prohibited." If this principle is enforced, then an intruder can not get access to some object just because the security administrator did not consider whether it should be restricted or not. Many members of the security community deem that if software were designed with more emphasis on security and if systems were configured properly, then there would be less security problems.

2.1.4 Approaches to Secure Computing

The following subsections, adapted from [Kemmerer 1994], presents four approaches to achieving secure computing, which may be used to implement an organisation's security policy. They are the use of special procedures for working with the system, the inclusion of additional functions or mechanisms in the system, the use of assurance techniques to increase one's confidence in the security of the system, and the use of intrusion detection systems.

(A) Procedural Approaches

Procedural approaches prescribe the appropriate behaviour for the user to follow when using the system. The periods processing approach for processing jobs at different security levels is an example of a procedural solution to satisfy a security requirement.

User guidelines for the appropriate choice of a password are the most prevalent example of using procedural approaches to achieve a security requirement. For example, to deter password guessing by a potential intruder one should choose a long password (at least eight characters) that is not obvious and not easily guessable (e.g., not a spouse's first name, a middle name, a login name, or any of these spelled backwards). In addition, a password should not be written down, or if it is it should not be written in an obvious place.

Another example of a procedural approach is a set of rules for the appropriate handling of removable storage devices. Oftentimes data that is perfectly safe while in a protected system is compromised by a penetrator getting access to removable storage, such as a dump tape, and analyzing it on an unprotected system. For this reason most security conscious organisations have explicit rules for handling removable media, such as requiring them to be stored in an approved vault.

(B) Functions and Mechanism

Including additional functions or mechanisms in a computer system is another way of enhancing computer security. The mechanisms presented in this section are grouped into authentication mechanisms, access control, and inference control.

Authentication Mechanisms

Authentication mechanisms are used to assure that a particular user is who he/she claims to be. One of the mechanisms is the *secure attention key*. This key, when hit by a user at a terminal, kills any process running at the terminal except the true system listener and thus guarantees a trusted path to the system. This will foil attempts at "spoofing," which is the process of fooling a user into believing that he/she is talking to the system, resulting in information being revealed.

Most of the password guidelines that were discussed above as a procedural approach can be enforced by the system. For instance, the password program can require long passwords and it can check the password chosen against an online dictionary or against a list of obvious passwords. The login program can also inform the user that it is time to change passwords and not allow further logins if the password is not changed in time. Finally, the system can generate secure passwords for the users using a secure password generator.

Other mechanisms include sophisticated authentication devices that depend on unique physiological or behavioral characteristic that can be examined for each user. The most common biometric devices are based on fingerprints, handprints, or retina patterns. The most common behavioral devices use voice, signature, or keystroke characteristics.

Access Control

Assuming that by using authentication mechanisms and good password practice the system can guarantee that users are who they claim to be, the next step is to provide a means of limiting a user's access to only those files that policy determines should be accessed. These controls are referred to as access control. Different applications and systems have different security requirements, and these requirements are expressed in the access control policy for the application or system. Access control policies are enforced by the access control mechanisms.

There are two types of access control:

- Discretionary access control - the owner of an object specifies what type of access the other users can have to the object. Thus, access is at the discretion of the owner. An example would be the use of passwords for file access whereby the owner selects a password for each file and distributes the password to those users to whom the owner wants to give access to the file. Another example that is used on Unix systems is the owner/group/other approach. The owner of a file assigns the types of access that are allowed for the owner, for all users in the group associated with the file, and for all users on the system.

- Mandatory access control - the system determines whether a user can access a resource based on the security attributes (e.g., labels or markings) of both the user and the object. Mandatory access control is often called non-discretionary access control.

A convenient way of describing access rights is with an access matrix. Each entry in the matrix is a set of access rights that indicate the access that the subject associated with the row has for the object associated with the column. Figure 2.2 is an example access matrix.

	OBJECTS				
SUBJECTS	01	02	03	04	05
S1	R		W	RW	W
S2		E		R	
S3		RW	E		
S4	RE		RW		RE

Figure 2.2: Example Access Matrix

There are two commonly used and more efficient ways of representing an access matrix in a computer system: access control lists and capability lists [Kemmerer 1994]. With the access control list approach each object has an access list associated with it. The capability list approach associates a list with each subject.

Inference Controls

The last class of security mechanisms discussed in this section is inference controls. These controls attempt to restrict database access to sensitive information about individuals while providing access to statistics about groups of individuals. The ideal is to have a statistical database that discloses no sensitive data.

As an example of the type of threat that is addressed by inference control mechanisms consider a database that contains enrollment and grade statistics for a university class. If Morgan is the only Economics major in a particular class one could deduce Morgan's grade by retrieving the average grade for the course, the average grade of all non-economics majors in the class, and the number of students in the class.

Two approaches to solving the inference problem are to restrict queries that reveal certain types of statistics and to add "noise" to the results returned. An example of a query restriction would be to restrict queries with more than some predetermined number of records in common or with too many attributes specified. As for adding "noise" to the statistical results returned, techniques used are systematic rounding, random rounding, and controlled rounding [Kemmerer 1994].

(C) Assurance Techniques

The third approach to enhancing the security of a system is to subject the system to rigorous assurance techniques that will raise one's confidence that the system will perform as desired. Among these techniques are penetration analysis, formal verification, and covert channel analysis. None of these methods guarantee a secure system. They only increase one's confidence in the security of the system.

Penetration Analysis

One approach to locating security flaws in computer systems is penetration analysis. This approach uses a collection of known flaws, generalizes these flaws, and tries to apply them to the system being analyzed. Usually a team of penetrators, called a tiger team, is given the task of trying to enter the system. Flaws in many major systems have been located by using this approach [Hebbard 1980, Linde 1975].

The problem with the tiger team approach is that like testing, "penetration teams prove the presence, not absence of protection failures" [Popek 1974]. This observation has led to the use of formal specification and verification techniques to increase ones confidence in the reliability and security of a computer system.

Formal Verification

Formal verification demonstrates that an implementation is consistent with its requirements. This task is approached by decomposing it into a number of easier problems. The critical requirements, which are usually a natural language statement of what is desired, are first stated in precise mathematical terms. This is known as the formal model or critical criteria for the system. For example, the formal model for a secure system could be that information at one security level does not flow to another security level. Next, a high level formal specification of the system is stated. This specification gives a precise mathematical description of the behavior of the system omitting all implementation details, such as resource limitations. This is followed by a series of less abstract specifications each of which implements the next higher level specification, but with more detail. Finally, the system is coded in a high order language.

This high order language implementation must be shown to be consistent with the original requirements.

The advent of the security kernel as a means of encapsulating all security relevant aspects of the system makes formal verification feasible. That is, by developing kernel architectures that minimize the amount and complexity of software involved in security decisions and enforcement, the chances of successfully verifying that the system meets its security requirements are greatly increased. Because the remainder of the system is written using the facilities provided by the kernel, only the kernel code must be verified. Examples of work in this area are [McCauley 1979, Walker 1980, Silverman 1983, and Bevier 1989]

Covert Channel Analysis

Covert channels signal information through system facilities not intended for data transfer, and they support this communication using methods not detected or regulated by the access control mechanisms. Storage channels transfer information using the manipulation of storage locations for their coding scheme. That is, the sending process alters some system attribute (e.g., a file lock or a device busy flag) and the receiving process monitors the alteration. For example, if two processes have only write access to a file, then the sending process could lock the file, and the receiving process could detect this change by attempting to write to the file. In this way, the receiver could interpret the file being locked as a one and it not being locked as a zero. Timing channels transfer information using the passing of time for their coding scheme; the sending process modulates the receiver's response time to detect a change in some shared entity.

Although there is concern that a user at a high security level may use a covert channel to signal information to a user at a lower level, the major threat from a covert channel is its potential to be employed by a Trojan horse. A *Trojan horse* is a program that gives the appearance of providing normal functionality, but whose execution results in undesired side effects.

In addressing the threat of covert channels two major challenges have been identified. The first challenge is in developing techniques to identify covert channels in a comprehensive, systematic manner. Several covert channel analysis techniques have been proposed and utilized. Usually these techniques base their analysis either on code inspection or on inspection of the high level specification. Among these techniques are the Non-Interference approach [Goguen 1982], the Shared Resource Matrix (SRM) methodology [Kemmerer 1983], and Information Flow analysis [Denning 1976]. The second, and more difficult challenge, is in removing the channels, or at least lowering their bandwidths, without rendering the system unacceptably slow or restrictive. References [Hu 1991, Karger 1991] provide excellent examples of how this second covert channel challenge is being met.

(D) Intrusion Detection and Prevention

Over the past decade, significant progress has been made toward the improvement of computer system security. Unfortunately, the undeniable reality remains that all computers are vulnerable to compromise. Even the most secure systems built today are vulnerable to authorized users who abuse their privileges. Given this reality, the need for user accountability is very important. Accountability is the key, both as a deterrent and for terminating abusive computer usage once it is discovered. In addition, the need to maintain a record of the transactions that have occurred on a system is crucial for performing damage assessment. In recognition of these needs and in recognition that security violations are a fact of life for computers, many systems implement a form of transaction record keeping called audit collection. The data collected is called an audit log. Additional systems and/or software are often required to generate the necessary audit data. For example, in order to produce audit records for Sun Microsystem's Solaris one needs to use the Basic Security Module (BSM) [Sun 1991]. The topic of intrusion detection is dealt with extensively in subsequent sections in this chapter.

2.1.5 Computer Security and Software Engineering

The key difference between secure software and other high quality software is that secure systems have to be able to withstand active attacks by potential penetrators. When developing any reliable software one must try to minimize the faults in the system and assure that accidental abnormal user actions or abnormal external events do not result in undesired events. When developing a secure system the developers must also assure that intentional abnormal actions cannot compromise the system. That is, secure systems must be able to avoid problems caused by malicious users with unlimited resources.

Because security is a system requirement just like performance, capability, and cost, it must be designed from the beginning. It must not be added on after-the-fact. In addition, because security is only one of many goals for a system, it may be necessary to trade off certain security requirements to achieve other goals, such as user friendliness.

When designers first start thinking about developing secure systems they often think that it would be beneficial to keep the system design secret. However, the security community realized many years ago that the benefits of an open design far outweigh any advantages of keeping the design hidden from would be intruders. The *open design principle* [Saltzer 1975] states that "the mechanism should not depend on the ignorance of potential attackers..." By having peer reviews of the design of a secure system, security weaknesses in the system are often discovered during the design phase. It is always better to have a colleague find a weakness during the design phase rather than having it discovered through a compromise after the system has been fielded.

The main difference between secure systems and other high quality systems is that secure systems are subject to malicious attack; therefore, it should be no surprise that a primary difference in developing secure systems is the need for additional testing and for a somewhat different form of testing. Penetration analysis is a form of testing. However, unlike standard testing techniques where a tester's primary task is to demonstrate that the program gives the correct result when presented with varying inputs, with penetration analysis the system is given input that is not expected to give a good result or operations are executed in an unexpected sequence. Thus, instead of concentrating on showing that the system gives the expected result when used properly, the testing concentrates on demonstrating that the system gives an undesired result when used improperly. Thus, the test cases concentrate more on trying the unusual or the unexpected.

Finally, because the need for a high level of assurance was recognized by the security community many years ago, the use of formal methods is more prevalent in the design and analysis of secure systems than in most other software development areas. All of the national and international evaluation criteria specify the need for the use of formal methods to achieve added assurance.

2.2 *Unix*

2.2.1 Definitions

Ironically, according to Seth T. Ross in his *Unix System Security Tools* textbook – "*There is no such thing as Unix*". In a nutshell, Unix, since its birth more than 30 years ago by AT&T, has metamorphosed millions of times. After the licensing of the AT&T source code in the early 1990s, each manufacturer produced its own version of the operating system resulting in thousands of Unix variants.

[Ross 2000] offers several practical definitions of Unix, they are as follows:

- *Legal.* While there may be no such thing as "Unix," the term is a trademark owned by the Open Group, an international consortium that demands the mark received proper attribution. Sometimes AT&T, Bell Labs, Novell, or X/Open Company Ltd. can be seen listed as the trademark holders -- it's been passed off time and again.
- *Technical.* According to the Unix FAQ, Unix is "an operating system typically written in C, with a hierarchical file system and integration of file and device I/O, whose system call interface includes services such as fork() and pipe(), and whose user interface includes tools such as cc , troff , grep , awk , and a choice of shell." It also provides a consistent approach to multitasking, with built-in operations for the creation, synchronization, and termination of processes. It is intrinsically portable between different kinds of computers.
- *Linguistic.* The name "Unix" was intended as a pun on the name Multics and was written "Unics" at first, for UNiplexed Information and Computing System. Both "Unix" and "UNIX" are in wide use today - "UNIX" isn't an acronym. This dissertation uses "Unix".

- *Social.* Many people who run Unix-like systems such as Linux think they're running Unix. Official Unix systems and unofficial Unix systems are commonly treated as belonging to a single category -- in books, in media coverage, on the net, and by general social consensus.

The following overview of the Unix operating system is adapted from [Bell Labs 2002].

2.2.2 An Overview of the Unix Operating System

The Unix operating system was designed to let a number of programmers access the computer at the same time and share its resources.

The operating system coordinates the use of the computer's resources, allowing one person, for example, to run a spell check program while another creates a document, lets another edit a document while another creates graphics, and lets another user format a document -- all at the same time, with each user oblivious to the activities of the others.

The operating system controls all of the commands from all of the keyboards and all of the data being generated, and permits each user to believe he or she is the only person working on the computer.

This real-time sharing of resources makes Unix one of the most powerful operating systems ever.

(A) The Uniqueness of Unix

Multitasking

Unix lets a computer do several things at once, such as printing out one file while the user edits another file. This is a major feature for users, since users don't have to wait for one application to end before starting another one.

Multiusers

The same design that permits multitasking permits multiple users to use the computer. The computer can take the commands of a number of users -- determined by the design of the computer -- to run programs, access files, and print documents at the same time.

System Portability

A major contribution of the Unix system was its portability, permitting it to move from one brand of computer to another with a minimum of code changes. At a time when different computer lines of the same vendor didn't talk to each other -- yet alone machines of multiple vendors -- that meant a great savings in both hardware and software upgrades.

It also meant that the operating system could be upgraded without having all the customer's data inputted again. And new versions of Unix were backward compatible with older versions, making it easier for companies to upgrade in an orderly manner.

Unix Tools

Unix comes with hundreds of programs that can be divided into two classes:

- **Integral utilities** that are absolutely necessary for the operation of the computer, such as the command interpreter, and
- **Tools** that aren't necessary for the operation of Unix but provide the user with additional capabilities, such as typesetting capabilities and e-mail.

Unix Communications

Unix e-mail, at first, permitted users on the same computer to communicate with each other via their terminals. Then users on different machines, even made by different vendors, were connected to support e-mail. And finally, Unix systems around the world were linked into a world wide web decades before the development of today's World Wide Web.

Applications Libraries

Unix as it is known today didn't just develop overnight. Nor were just a few people responsible for its growth. As soon as it moved from Bell Labs into the universities, every computer programmer worth his or her own salt started developing programs for Unix.

Today there are hundreds of Unix applications that can be purchased from third-party vendors, in addition to the applications that come with Unix.

(B) How Unix is Organized

The Unix system is functionally organized at three levels:

- The kernel, which schedules tasks and manages storage;
- The shell, which connects and interprets users' commands, calls programs from memory, and executes them; and
- The tools and applications that offer additional functionality to the operating system

The Kernel

The heart of the operating system, the kernel controls the hardware and turns part of the system on and off at the programmer's command. If the computer is asked to list (*ls*) all the files in a directory, the kernel tells the computer to read all the files in that directory from the disk and display them on the screen.

The Shell

There are several types of shell, most notably the command driven Bourne Shell and the C Shell, and menu-driven shells that make it easier for beginners to use. Whatever shell is used, its purpose remains the same -- to act as an interpreter between the user and the computer.

The shell also provides the functionality of "pipes," whereby a number of commands can be linked together by a user, permitting the output of one program to become the input to another program.

Tools and Applications

There are hundreds of tools available to Unix users, however, some have been written by third party vendors for specific applications. Typically, tools are grouped into categories for certain functions, such as word processing, business applications, or programming.

2.3 Security in Unix

Dennis Ritchie, one of the creators of Unix, wrote on the security of the system: "It was not designed from the start to be secure. It was designed with the necessary characteristics to make security serviceable." Unix was never well-known for its security. However, as with any other operating system, Unix security can be enhanced by applying to the security approaches, functions and mechanisms, assurance techniques and intrusion detection and prevention, discussed in the section 2.1.4.

Nevertheless, Unix offers basic security and protection in the following ways [Adfa 1995]:

- Protection between processes. Unless shared memory or tracing a child program is used, a user can't look at the memory of another process. Users can only send signals and get the status of their own child processes.
- Protection between users. Users need a password to log in and use a Unix system.
- Protection at the file level. Remember that in Unix, nearly everything is a file. Unix allows three levels of protection: the user-level, the group-level, and everybody else. For each level, there are three file privileges: a file is readable, a file is writable and a file is executable.

Most of the Unix security features are done at the file and/or user level. Users can't access most people's directories because they have been denied permission, and many of the programs on the system are similarly protected against users.

Very few of the Unix security features are provided at the system call level. That is to say, very few of the system calls have anything to do with security, but all of the system calls have their arguments and results checked by the operating system to ensure that users are not trying to do anything they are not allowed to.

Remember that the basic access to an operating system's services is through the system calls. If these system calls do not check if the user or process making the call has the authority, there is no security. Fortunately, nearly all Unix systems calls are now very secure; there were a few system call *holes* in earlier systems, but these have been fixed. But as new system calls are introduced, new security holes appear. Security holes in general have occurred in poorly written Unix system and application programs.

2.3.1 Unix Security Issues

(A) Identification and Authentication

When a user logs into his/her account the system carries out any commands that it finds in special "login" files. These commands define the user's working environment: which editor to use, which printer to send work to and where to deliver printed output to. Making changes to these login files changes the user's working environment.

Once a user is logged into his/her account the user is always placed in his/her home directory. This is the user's very own location in the file system: it never varies. Any Unix system can have many users on it at any one time. User's home directories are usually grouped together under a system directory such as /home. A large Unix system may have several hundred users, with their home directories grouped in subdirectories according to some schema such as their organisational department.

There may be several Unix systems at a site connected together by a network. If the user has accounts on these remote systems the user can login to them from the system he/she is currently using. Distributed applications are left on their own for client/server authentication. Many applications are distributed by using programs such as telnet and ftp, which rely on passwords sent in the clear. Two ways of doing this are listed below:

Logging in with rlogin (remote login)

Logging in with telnet

(B) Access Control

Unix provides access control based on object permissions. The object permissions are discretionary access controls. Most objects are accessible through the file system in

Unix, and the access control mechanisms are implemented as a part of the file system. Some interprocess communication mechanisms are not represented in the file system and have their own access control mechanisms.

Unix provides access control using permission bits. The permission bits provide a fixed granularity of user (really owner), group, and other. This is the same as having an access control list that can only have three rules in it: one for the owner, one for a single group, and one for everyone else. Groups are provided for ease of administration. Groups are defined by the system administrator and are the only mechanism for specifying permissions for more than a single user.

Unix doesn't have a generic mechanism to give users selected system privileges. The root user, or superuser, has all system privileges. Administrators using the setuid and group features can build privilege mechanisms. *Setuid* is a powerful Unix feature that allows one user to assume the identity of another while executing a file (often this user id is zero or root--the administrative superuser). These applications are privileged no matter who invokes them, and as such must be as verified as secure as any part of the system trusted path. Much security breaches in Unix implementations are due to errors in coding these programs. The end effect could be that if a privileged application executes a malicious program, an entire system and network could be compromised.

There are three types of permissions:

r read the file or directory

w write to the file or directory

x execute the file or search the directory

Each of these permissions can be set for any one of three types of users:

u the user who owns the file

g members of the group to which the owner belongs

o all other users

The access permissions for all three types of user can be given as a string of nine characters:

```
user  group  others
r w x  r w x  r w x
```

(C) Accountability

Unix tightly binds a User ID (UID) that uniquely identifies a user on that host to each process or entity that performs an action on the system. The UID name space can be expanded to a collection of hosts (domain) with the Network Information Service (NIS). The UID is part of the user block (ublock) of information associated with every process connected with a user's login. This information is available to audit mechanisms.

(D) Audit

There is no generic auditing capability in Unix. Authentication related events are logged, but in event specific files such as *sulog* or privilege escalation, and *utmp*, *wtmp*, and *loginlog* for login information.

The *syslog* logging facility in Unix systems is focused on error logging and isn't designed for security purposes.

(E) Integrity

Unix uses its file permissions to prevent access to the operating system itself. This is the only Unix interface or facility for integrity services, except for explicit use of files system structure checks (*fsck*) and dump/restore. The sum command can be used for some purposes, but it is not a cryptographically strong checksum.

(F) Reliability of Service

When a user enters a command it invokes a program. While this program is running it is called a process. It is important to grasp that although there is only one copy of a program held in the file system, any number of processes can be invoked which run this program. When the operating system is started after a boot, a single process is started. This process is the parent of all subsequent processes. Each process created on the system has a unique number, known as its PID, associated with it.

When a user logs into the system a process is started to run the user's shell program. Any processes that are started from within the shell - such as entering a command - are the children of this process. A process can have many children, but only one parent.

(G) Data Exchange

If data is important, it may need to be encrypted, especially if it is to be sent over a network. The DES encryption algorithm is common in;the US, although it is illegal to export it. It is also suspected that the American Security Agencies are able to crack it. There are a number of "public key" encryption algorithms which are very secure. The PGP (Pretty Good Privacy) algorithm has a version that is legal to use in Australia. It allows encryption and digital signatures.

2.3.2 Enhancing Unix Security

Third party tools and techniques are available to help the system administrator and system programmer to secure Unix. This section, compiled from [Smith 1993] includes a selection of them with a discussion on what they do. Whilst these tools do not prevent software vulnerabilities, they may help detect any intrusions that may occur through the exploitation of those vulnerabilities, or prevent the use of network sniffers to capture important authentication data.

(A) Cryptographic Tools

Kerberos

The following analysis is drawn from [Stevens 1990], [Bellovin 1991] and [Kohl 1990].

The Kerberos authentication system was produced at MIT as a part of Project Athena. It is a system that uses protocols which allow authentication to take place, even under the assumption that the network is under the complete control of an enemy. Kerberos uses a private key cryptosystem to protect the information from disclosure and modification. The user interface is the same as that for normal passwords.

The major strength of Kerberos is that the password is never transmitted on the network in plain text. This reduces the likelihood of the password being captured and replayed. The tickets and authenticators include a timestamp which aids in preventing replay attacks (where an intruder replays a valid authentication sequence).

This style of authentication was designed with the distributed or networking environment in mind. It is well suited to the client-server model often used in networking applications. Since both the ticket and authenticator contain the network address of the client, another workstation cannot use stolen copies without changing their network address.

Kerberos contains a number of minor deficiencies which should be well understood in order to use it effectively. Installing Kerberos will increase the level of security over normal passwords, provided its limitations are understood and accepted.

DES

One of the most widely used encryption systems today is the Data Encryption Standard (DES) developed in the 1970s by IBM [FIP 1977], [Caelli 1991]. DES is a bit permutation, substitution, and recombination function performed on blocks of 64 bits of data and 56 bits of key (8 characters of 7 bits). The algorithm is structured in such a way that changing any bit in the input has a major effect on almost all of the output bits.

The DES algorithm can be used in four modes:

- Electronic Code Book (ECB);
- Cipher Block Chaining (CBC);
- Output Feedback (OFB);
- Cipher Feedback (CFB).

Each mode has particular advantages in some circumstances, such as transmitting data over a noisy channel, or when it is necessary to decrypt only a portion of a file.

DES uses the same key to encrypt the data and decrypt the data. Therefore, it is essential to use techniques that keep the secrecy of this key intact. Practical experience of using DES in a global situation highlights the difficulty of using DES in groups where keys must be distributed regularly to differing time zones. Poor key management leads to the reduced effectiveness of DES.

MD2, MD4, MD5

The MD2 Message Digest Algorithm [Rivest 1992] was designed to exploit to 32-bit RISC architectures to maximise its throughput, and does not require large substitution tables. The MD5 Message Digest Algorithm [Rivest 1992a] is a proposed data authentication standard. MD5 attempts to address potential security risks found in the speedier but less secure MD4.

The message digest algorithms generate a 128-bit signature (fingerprint or message digest) from a given block of text. The signature is designed to prevent someone from determining a valid block of text from a given signature or to modify a block of text while keeping the same signature.

(B) Security Assessment Tools

Tripwire

Tripwire is a file integrity checker using a number of cryptographic checksumming algorithms in parallel for added security [Kim 1992].

Tripwire makes use of several message digesting algorithms. These are:

- MD5
- MD4
- MD2
- Snefru
- CRC-3

- CRC-16

The use of more than one of these algorithms in parallel greatly decreases the chances of an intruder being able to modify a monitored file without detection. Initially, a reference database is built, immediately after the installation of the operating system and any products, and prior to reconnecting to the network. This way, one can be sure that the files have not been modified by an intruder. The output of Tripwire (as well as Tripwire itself) should be kept on a hardware write protected disk to prevent modification (a read-only mounted partition is not sufficient as this may be remounted read-write by the intruder). Tripwire should then be run at regular intervals to verify the integrity of key system files. Another alternative to using hardware protected media is to print out a copy of Tripwire's results. An intruder must gain physical access to the premises to adjust the original data from Tripwire. This helps if there is any suspicion on the integrity of the Tripwire database.

It is meaningless to use Tripwire to protect a file such as the system password file as users have the ability to change their password at any time, and thus the file checksums will also change.

(C) Security Enhancement Tools

TCP Wrapper

TCP Wrapper (also known as LOG TCP) is a package that is used to monitor incoming IP connections, log them, and provide a number of add-on services including a limited form of access control and some sanity checks [Venema 1992].

TCP Wrapper is an extremely simple, and yet effective tool. It is very useful in preventing connections from outside an organisation from approaching the systems. It is possible to allow certain connections (for example, mail) to the systems, while restricting others. Even if an intruder learns an account and password for the system, they must first penetrate a "trusted" system before they can gain access to the system [Curry 1990]. It is therefore imperative that users do not use the same password on all systems.

The TCP Wrapper, when properly configured, will reduce a system's exposure to intruders, and hence reduce their ability to compromise the security of a system by exploiting software vulnerabilities.

Token Generators

Token Generators are hardware packages that implement password "tokens" or one-time use passwords [Brand 1990], [CER 1992], [Ellison 1992]. Token generators are implemented using a variety of schemes.

One system operates by challenging the user with a seven digit number (in phone number format). A PIN number and the challenge number are entered into the hand held device, and it gives a seven-digit response code to reply with.

Other systems use a changing, non-reusable password system. Each time a user authenticates, a new password is supplied by the hand held device. There is no challenge-response system, and the user must keep in synchronisation with password usage to prevent a Denial of Service. Some systems can support single use password generation for up to eight separate host systems. Some of these systems require the user to enter a PIN before the next password is issued.

Another system displays the password continuously, changing it every minute or so. The host must not only keep the user's key (for generating the same sequence), but also a synchronised clock.

The one-time password system is extremely effective in preventing replay attacks, provided the enemy does not know the sequence of generated passwords (either by guessing, or possession of a similar device and key).

One of the major disadvantages is that to authenticate, a user must carry the hardware with them at all times. If they do not possess their hand held device, then authentication cannot take place.

S/Key

S/Key is a software system designed to implement a secure one-time password scheme [Karn 1993]. It uses 64 bits of information transformed by the MD4 message digest algorithm [Rivest 1992]. The 64 bits are supplied by the user in the form of six English words that are generated by a secure computer. Ultimately, this computer could be a pocket sized smart card, a standalone PC or Macintosh, or a secured machine at work.

2.3.3 Similarities and Differences Between Windows NT and Unix

Unix and Windows NT security features are compared to each other in this section. Table 2.1 provides a synopsis of each operating system's compliance in each of the seven key operating system security areas.

Table 2.1: Similarities and Differences Between Windows NT and Unix

Key OS Security Areas	Windows NT	Unix
Identification/ Authentication	• required before system access • trusted path • username/password as default • can encrypt password file • *GINA available to add mechanisms • distributed application authentication	• required before system access • no trusted path • username/password as default • can hide password file • *ad hoc addition of third party mechanisms • *Kerberos add-on
Access Control	• Discretionary access controls • implemented in Security • Reference Monitor • Access Control Lists • 27 assignable user rights	• Discretionary access controls • Implemented in file system and other kernel locations • User/Group/Other mode bits • *superuser* and *setuid* mechanisms
Accountability	• Security ID • Maintained across client-server	• User ID • Maintained across client-server in certain applications
Auditing	• Event Logging ○ Single log file ○ Comprehensive set of events	• Multiple event specific logs • Limited set of events • Add-on packages for comprehensive security auditing
Integrity	• ACLs protect operating system • Authenticode for verifying code • CAPI for ápplications • Integral file system utilities for availability	• Mode bits protect OS • no mechanism to verify • code • GSS-API for applications • *add-on file system utilities for availability
Confidentiality	• CAPI for applications • Secure RPC • PPTP and SSL • *IPSEC add-on	• *GSS-API add-ons for applications • secure RPC • SSL toolkits • *IPSEC add-on

Manageability	• Low security by default • Single tool set, consistent GUI with other management tools • Low skill level required	• Low security by default • Multiple vendor specific tool sets, not consistent with other management tools • High skill level required
Security Partitioning	• Security domains separated • Separate user and kernel address space • Individual process address space • Single module implementation • Objects cleared prior to re-use	• Security domains separated • Separate user and kernel address space • Individual process address space • Distributed implementation • Objects cleared prior to re-use

* This denotes that the basic operating system does not come with this feature out of the box. They are add-on software utilities or applications provided by third-party vendors.

2.4 *Intrusion Detection*

2.4.1 What is Intrusion Detection?

"Intrusion detection is the *process* of *identifying* and *responding* to *malicious activity* targeted at *computing and networking resources*. " [Amoroso 1999]

Intrusion detection is foremost a '*process*' involving technology, people, tools, time and interaction between entities. It is not a plug-and-play black box solution. The identification ('*identifying*') of an intrusion can be done before, during, or after the target malicious activity proceeds. If the identification of an intrusion is done after-the-fact, then the question of damages incurred and how such intrusions occurred must be addressed. Response ('*responding*') follows after identification of the intrusion. Meanwhile, '*malicious activity*' refers to security-relevant actions that are harmful. '*Computing and networking resources*' is the context of security referred to in this thesis because the primary interest here is the security of computers in contrast to household (e.g. use of surveillance cameras, and the like, to detect intruders) security.

2.4.2 Why Intrusion Detection?

Standard security measures like firewalls and virtual private networks (VPNs) are insufficient. Hacker attacks in the form of email based Trojan horses, stealth scanning techniques and actual attacks can bypass firewalls. There are also hacking techniques that exploit applications that are allowed to pass through the firewalls in order to enter the target system. Intrusion detection proposes a solution to the weaknesses exposed in the standard security measure by warding off attempted unauthorized access and deny access to would-be intruders.

2.4.3 The Evolution of Intrusion Detection: An Overview

Based on [Innella 2001] and [Bruneau 2001], intrusion detection has evolved from labour-intensive manual scrutiny of audit logs to fully automated intrusion prevention systems. Although the concept of intrusion detection has been around for decades now, it only gained popularity very recently.

In the late 70s, administrators used to print audit logs, that were used as a forensic tool, that were stacked four to five feet high by the end of the week. This method of investigation was time-consuming and only detects the cause of security incidents after the fact. Soon after, the notion of intrusion detection surfaced with the publishing of James Anderson's paper, Computer Security Threat Monitoring and Surveillance in 1980. After that, several key events in IDS technology have advanced intrusion detection to its present state.

Before long Stanford Research Institute (SRI) developed a means of tracking and analyzing audit data on the ARPANET in 1984. Shortly after, this institute completed a Navy SPAWAR contract with the first functional intrusion detection system, IDES. Then Dr. Denning from SRI published the pivotal work, An Intrusion Detection Model that revealed the essential information for commercial IDS development. This paper is the basis for most of the work in IDS that followed. Then in 1988, the Haystack project at Lawrence Livermore Labs at UC Davis released another version of IDS for the US Air Force (USAF) that analyzed audit data by comparing it with defined patterns.

Subsequently, an iteration of the IDS for the USAF, called Distributed Intrusion Detection System (DIDS) was built. DIDS augmented the existing system by tracking client machines as well as the servers it originally monitored. In 1989, the Haystacks Labs released Stalker, a host-based, pattern-matching system that included robust search facilities to query audit records. The Haystack progressions, coupled with SRI's developments, significantly advanced host-based ID technologies.

Network intrusion detection system first came into recognition with UC Davis's Todd Heberlain's development of Network Security Monitor (NSM). Heberlain's NSM was deployed at major government implementations and also extended to the DIDS project, where along with the Haystack team introduced concept of hybrid intrusion detection. This fusion revolutionized the IDS subject and boosted it into the commercial scene.

The 1990s saw the advent of commercially driven IDS developments. Haystack Labs with its Stalker line of host-based products, Science Applications International Corporation (SAIC) with a form of host-based intrusion detection, called Computer

Misuse Detection System (CMDS), and the Air Force's Cryptologic Support Center developed the Automated Security Measurement System (ASIM) to monitor network traffic on the US Air Force's network. ASIM was the first solution to incorporate both a hardware and software solution to network intrusion detection. ASIM project formed a commercial company in 1994, the Wheel Group. Their product, NetRanger, was the first commercially viable network intrusion detection device.

The intrusion detection market began to gain in popularity and truly generate revenues around 1997. In that year, the security market leader, Internet Security Systems (ISS), developed a network intrusion detection system called RealSecure. In 1998, Cisco Systems bought over the Wheel Group, attaining a security solution for their customers. Similarly, the first visible host-based intrusion detection company, Centrax Corporation, emerged as a result of a merger of the development staff from Haystack Labs and the departure of the CMDS team from SAIC. At present, market statistics indicate that IDS is amongst the top selling security vendor technologies and should proceed to increase. Moreover, US government initiatives, such as the Federal Intrusion Detection Network (FIDNet), are also adding thrust to the advancement of IDS.

Thus far, there's application intrusion detection, integration of artificial intelligence, heuristics and rules-based intrusion detection, and the like. Regardless of how intrusion detection technology evolves, it is now an indispensable component of information and computing security. Figure 2.3 depicts the evolution of intrusion detection systems over the past few decades.

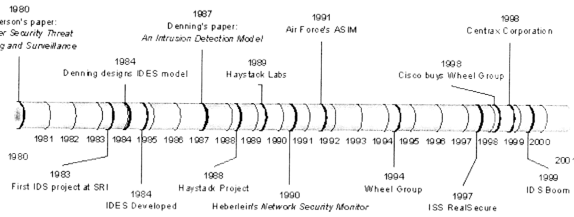


Figure 2.3: The Evolution of Intrusion Detection Systems

2.4.4 Technology Overview of Intrusion Detection Systems

Intrusion detection systems can typically be divided into two main types, network-based or host-based. Network-based IDSs look for attacks in the network traffic while host-based IDSs look for attacks in log files. Although recent advances in technology has given birth to commercially-driven hybrid IDSs that incorporate both network-based and host-based IDS technologies, they remain the more expensive solutions.

(A) Network-Based Intrusion Detection

Network-based IDSs draw on unprocessed network packets as the data source. A network-based IDS usually utilizes a network adapter running in promiscuous mode to monitor and analyze all traffic in real-time as it traverses the network. Its attack recognition module uses four common techniques to recognize an attack signature:

- Pattern, expression or bytecode matching
- Frequency or threshold crossing

- Correlation of lesser events
- Statistical anomaly detection

Upon detecting an attack, the IDS's response module can notify, alert, react, or record the break-in session in response to the attack.

Pros and Cons of Network-Based Intrusion Detection Systems

Table 2.2 describes the pros and cons of network-based intrusion detection systems.

Table 2.2: Pros and Cons of Network-Based Intrusion Detection Systems

Pros of Network-Based IDS	Cons of Network-Based IDS
• Lower cost of ownership • Detects attacks that host-based systems miss • More difficult for an attacker to remove evidence • Real-time detection and response • Detects unsuccessful attacks and malicious intent • Operating system independence	• Overloading can occur due to high-volume network traffic • Limited visibility on components other than the network • False alarm reporting • Excessive management for IT staff • "Blind" IDS sensors may allow encrypted hazardous code to pass through

(B) Host-Based Intrusion Detection

Host-based intrusion detection emerged in the early 80s before networks were as prevalent as they are today. Host-based IDS typically monitor system, event, and security logs on Windows NT and syslog in Unix environments. When any of these files change, the IDS compare the new log entry with attack signatures to see if there is a match. If so, the system responds with administrator alerts and other calls to action.

Host-based IDS have augmented to incorporate other technologies such as inspection of key system files and executables using checksums at regular intervals for unanticipated alterations. The timeliness of the response corresponds to the frequency of the polling interval. Lastly, some products monitor port activity and alert administrators upon access of particular ports. This type of detection introduces a basic level of network-based intrusion detection.

Pros and Cons of Host-Based Intrusion Detection Systems

Table 2.3 describes the pros and cons of host-based intrusion detection systems.

Table 2.3: Pros and Cons of Host-Based Intrusion Detection Systems

Pros of Host-Based IDS	Cons of Host-Based IDS
• Verifies success or failure of an attack • Monitors specific system activities • Detects attacks that networked-based system miss • Well-suited for encrypted and switched environments • Near-real-time detection and response • Requires no additional hardware • Lower cost of entry	• No real-time intrusion detection • Defenseless against unknown attack signature models • Reactive IDS, when working with firewalls, are exposed to insiders • High-incidence reporting or false alarms • Once the system reacts to an attack, the damage is done • Negative audit trails record only unsuccessful attempts to access data

(C) Hybrid Intrusion Detection Systems

Hybrid intrusion detection systems offer management of and alert notification from both network and host-based intrusion detection devices. Hybrid solutions provide the logical complement to network-based ID and host-based ID - central intrusion detection management.

2.4.5 Methods Used in Intrusion Detection Systems

Methods vary among intrusion detection; however, given an IDS, a collection of common methods can be identified. The five techniques presented here are from [Amoroso 1999]; they are as follows:

(A) Audit Trail Processing

Audit trail processing is the most common ID technique. Figure 2.4 presents a high-level depiction of audit trail processing.

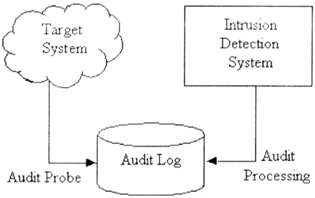


Figure 2.4: High Level Depiction of Audit Trail Processing

The idea in an audit trail processing is that an existing log is available for parsing and interpretation by an IDS. This method is typically performed off-line and rarely involves any real-time analysis. Such processing introduces issues of audit trail formats, storage techniques, archival policies, and real time auditing standards. It also raises issues in an intrusion-processing center of the manual processes required by human beings involved in the audit trail processing, particularly for archived data. Unfortunately, the security community has been negligent in the area of developing standard formats and techniques for audit trails.

(B) On-the-Fly Processing

This method, sometimes called network intrusion detection, involves the monitoring of traffic so that real-time or near real-time analysis can be done with respect to appropriate detection algorithms. Specified strings of characters are often used in this method as a means for parsing traffic for so-called "dirty words". These dirty words might include '/etc/password/', '/etc/shadow', or other sequences one might consider suspicious to be passing through an IDS. Figure 2.5 presents a high-level depiction of on-the-fly processing.

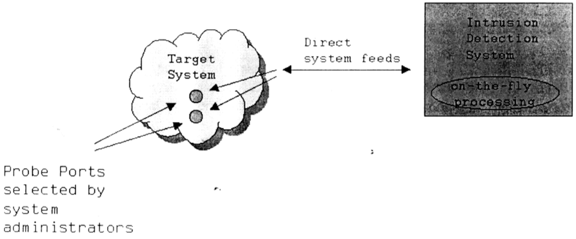

Target System
Direct system feeds
Intrusion Detection System
on-the-fly processing
Probe Ports selected by system administrators

Figure 2.5: High Level Depiction of On-the-Fly Processing

(C) Profiles of Normal Behaviour

Profiles of normal behaviour are used in intrusion detection to capture expectations about user and system computing and networking activity. This follows the basic paradigm of comparing expectations about behaviour with actual observations. The creation of such profiles involves at least the following three basic concerns—it involves much more, but these are the important ones.

- *Estimation of Initial Profiles.* Initial profiling for new users and systems requires estimation of expected behaviour. Such estimation is nontrivial and makes profiling vulnerable to malicious teaching approaches by intruders (i.e., by altering initial behaviour to set-up an intrusion profile to support a subsequent attack). One way to deal with such initial profiling problems is to maintain a stealth intrusion detection system so that new users are not aware that their behaviour is being profiled. This raises all sorts of legal, ethical and organisational policy issues that may make such a method unacceptable. This initial state problem may become less serious as more empirical evidence emerges for a given monitored environment.
- *Fine-Tuning of Profiles.* Observed user and system behaviour provides a basis for fine-tuning existing profiles. Proper fine-tuning is also nontrivial, as it requires attention to statistical concerns about probability of occurrences and regularity of events. In the best case, fine-tuning would be automated, but manual techniques must be developed for this before such automation can be considered trusted. This fine-tuning is also vulnerable to malicious teaching approaches by profiled users or systems.
- *Profiling Using All-Source Information.* Information should be used from any relevant source to more accurately predict expected behaviour. These sources do not have to be based on computing or networking information. In fact, some of the most powerful profiling information is derived from sources that are out of band with

respect to any computer or network (e.g., personal characteristics and habits). Figure 2.6 depicts these IDS profiling concerns.

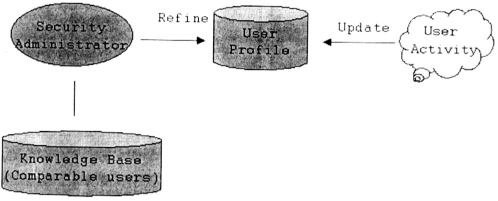


Figure 2.6: Profiling Normal Behaviour

(D) Signatures of Abnormal Behavior

The use of abnormal behaviour signatures, also called attack signatures in some security books and research articles, is particularly common in on-the-fly IDS. These abnormal behaviour signatures generally come in one or two different flavours:

- *Known Attack Descriptions*. Dynamic descriptions of related activity patterns that might constitute a security problem. These descriptions of known attacks are often referred to as attack signatures. Databases of these descriptions are reminiscent of virus databases in virus detection software.

  Suspicious String Patterns. Designated character strings (like '/etc/shadow', 'top secret', or 'proprietary') that correspond to traffic content that must be considered suspicious. Security administrators often determine these locally.

Figure 2.7 depicts the use of abnormal behaviour signatures for intrusion detection.

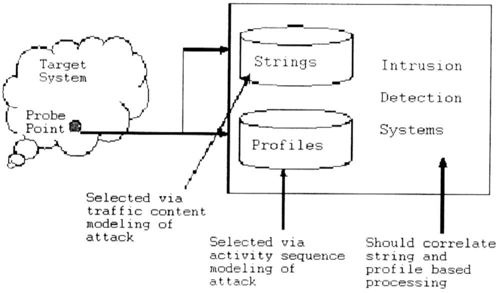
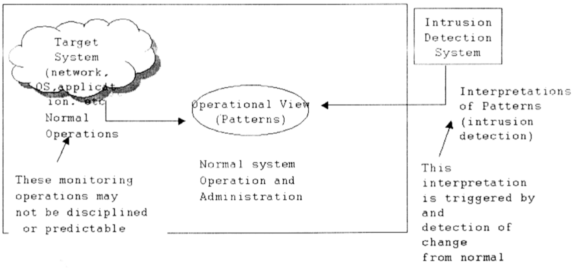

Target System
Probe Point
Strings
Profiles
Intrusion Detection Systems
Selected via traffic content modeling of attack
Selected via activity sequence modeling of attack
Should correlate string and profile based processing

Figure 2.7: Abnormal Behaviour Signature Method

(E) Parameter Pattern Matching

This method involves the use of day-to-day operational experience as the basis for detecting anomalies. From a logical perspective, this method can be viewed as a special case of the normal profiling method. It is separated out here because the explicit development of user and system security profiles may not be included in the approach. Instead, operators doing normal system and network management activity might, or might not, detect some sort of change in the parameters they typically monitor – hence our use of pattern matching term for this method. Actually, one of the more attractive characteristics of this method is that the administrators are not specifically targeting security issues. This introduces a more robust environment in which anomalies and patterns might be detected and matched.

Such pattern matching constitutes an especially powerful processing approach because it provides an intrusion detection capability for attacks that might not be predictable. In fact, human operators in a network operations center might detect subtle changes as part of their normal security and network management operations that they can neither explain nor understand. It is these types of unpredictable changes, however, that could lead to detection of a problem. Figure 2.8 depicts a high-level view of the pattern matching method.



Target System (network, OS, application, etc)
Normal Operations
These monitoring operations may not be disciplined or predictable
Operational View (Patterns)
Normal system Operation and Administration
Intrusion Detection System
Interpretations of Patterns (intrusion detection)
This interpretation is triggered by and detection of change from normal

Figure 2.8: Pattern Matching Method of Intrusion Detection

2.4.6 Organisation of Intrusion Detection Systems

According to [Amoroso 1999] the primary components of a typical IDS can be organized into a schema not unlike the one depicted in Figure 2.9.

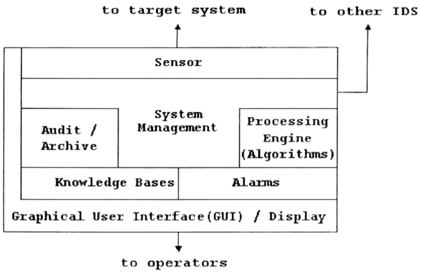


Figure 2.9: Intrusion Detection System Components

Here, a brief outline of each intrusion detection component is described.

- *Sensor*. This component provides the necessary information about the system targeted for ID. Sensing components are also sometimes referred to as Event-Boxes or E-Boxes. It is not uncommon for sensors to be physically remote from the rest of the intrusion detection system.
- *System Management*. Every IDS needs some system management function to maintain control over the internal components and to provide a means for communications with other IDSs. This management can be centralized or distributed in the system architecture. Network management systems based on the Simple Network Management Protocol (SNMP) and the Remote Monitoring (RMON) Management Information Base (MIB) are becoming more and more common as the basic system management functionality for IDSs.

- *Processing Engine (algorithms).* Processing is obviously required for ID and the associated processing algorithms are generally nontrivial. Clearly, at the highest level, we can characterize ID processing as consisting of a collection of different goals including reduction of irrelevant data, identification of key intrusion evidence, decision-making about evidence with respect to defined thresholds, mining of intrusion data warehouses for specific patterns or trends, and decision-making about the types of response activities to initiate. Practical concerns such as amounts of available processing power and memory inevitably arise in this area.
- *Knowledge bases.* Knowledge bases in IDSs come in a collection of different flavours. They provide a means for profiling of users and systems, for capturing attack signatures to be used in detection, and for keeping any information that may be considered useful in the processing, correlation, and analysis of potential intrusion activity. An important research issue in knowledge bases involves the development of common specification means for encoding profile and attack information.
- *Audit/archive.* Proper storage of target system activity in audit logs and archives requires considerable thought about the length of time to keep information, the manner in which the information should be protected, and the formats in which the information should be encoded or encrypted for storage and retrieval. Existing tools that generate audit trails are not presently well suited for integration with ID tools. This is an area in which vendors must do a better job in the future.

Alarms. In current state-of-the-practice IDSs, alarms are somewhat monolithic in the sense that they typically just alert a human being via a message, email, or page. As IDS technology matures, alarms must evolve into more dynamic directives from

- Check for unauthorized services. Inspect */etc/inetd.conf* for unauthorized additions or changes. This file contains the list of servers that *inetd* invokes when it receives an Internet request over a socket (i.e. telnet, ftp, etc). In particular, search for and check all entries that execute a shell program (for example, */bin/sh* or */bin/tcsh*).

2.6 Summary

This background chapter provided a literature review on Unix-based intrusion detection. It began with a discussion of computer security in general and proceeded to elaborate on the Unix operating system. It then discussed the application of security in Unix and made a comparison between Unix and Windows NT in seven key operating system security areas. This was followed by an elaboration on the topic of intrusion detection. Finally, the subject of intrusion detection in Unix was addressed.

resources is often much greater than the external threat, the use of ID on internal choke points should increase dramatically.

The major functional components included in an IDS architecture can be derived directly from the Figure 2.10. The components are as follows:

- *Target system.* The system on which activity and behaviour is being examined from an ID perspective is considered the target system. Clearly, the target system must have assets worth protecting before any IDS can even be considered. Corporate Intranets are good examples of typical target systems for intrusion detection. As the United States National Information Infrastructure (NII) becomes more dependent on computing and networking, it too becomes a good candidate for ID.
- *Feed.* The abstract notion of "feed" is meant to represent some means for deriving information from a target system for IDS processing and analysis. Obviously, in a network setting, this feed might constitute a live, dedicated network connection in the traditional sense; but it could mean a flow of information through any number of interim storage points and processing components. A key issue with respect to feeds is whether to carry monitored traffic from a sensor to a central correlation site for processing or to perform processing locally and carry alarms back to the central correlation site for processing. ID experts should be able to rattle off the pros and cons of distributed versus centralized feed processing feed processing (see below).
- *Processing.* We refer to processing as the execution of algorithms designed to detect malicious activity to some target system. These algorithms will typically make use of basic principles and heuristics to direct the types of processing performed.

Furthermore, the underlying physical architecture associated with processing must have sufficient power and storage to implement the selected algorithms.

- *Knowledge base*. In an IDS, knowledge bases are used to store information about attacks as signatures and strings, user and system behaviour as profiles, and other related information. These knowledge bases must be designed with appropriate protection, capacity, and processing queries to support the desired ID functions. They also must be associated with an update function so that information about new attacks can be installed into existing knowledge bases.
- *Storage*. The types of information that must be stored in an IDS will vary from short-term cached information about an on-going session to longer term event-related session information that must be sent to an audit trail or archive. Storage capacity requirements will obviously be environment specific. A key issue here is that as network capacities grow from slow Ethernet to much greater rates, short term cached information will require memory. Hardware packet capture routines will also greatly increase the need for interim storage as target system capacities increase.
- *Alarms/directives*. The most familiar response in an IDS is to send an alarm to an interested party, processing component, or other IDS. This may or may not be enough in a given target environment. For example, the inclusion of alarms that automatically reconfigure parts of the ID based on detected activity are becoming more feasible. As this trend continues, we believe that IDSs will require messaging architectures for transmitting information between components.
- *GUI/operator interface*. Display (graphical or otherwise) by an IDS for an operator or administrator requires attention to proper presentation, combination, and

representation of information. Most commercial IDSs are migrating to a point and click interface for administration and interpretation, which may be viewed as bad news by hard core Unix administrators. In reality, operational environments only achieve a proper user interface after considerable live experience and feedback to and from interface developers and the operational users. Few ID researchers and system vendors understand this subtle point. Instead, they continue to point to a glitzy user interface as a salient feature of their system.

- *Communications infrastructure*. Different components of an IDS and different IDSs require a means for communication. This may involve infrastructure protections such as an encryption and access control to protect information such as alarms in transit between components to ensure that information or requests from one part of the ID infrastructure get delivered reliably to the appropriate destination. For the average Intranet with multiple ID probe points, the communication infrastructure is the actual Intranet, perhaps with virtual private network (VPN) transport. Alternatives include dial access transport for low volumes or frame relay private virtual circuits (PVCs) for greater volumes.
- *Multiple IDSs*. As was suggested, some environments may involve the use of more than one intrusion detection system. When this is true, the overall infrastructure must support the types of care and maintenance required to deal with multiple systems. Efforts such as the Common Intrusion Detection Framework (CIDF) are being organized to ease the use of multiple systems from different vendors. The US National Information Infrastructure (NII) is a classic example of a collection of resources requiring multiple IDSs.

- *Network management.* Network management systems are commonly found in any nontrivial environment performing ID. The network management system may be embedded into the intrusion processing or it could be used to complement ID via remote monitoring and administration activity. Most IDS researchers forget this interaction between ID and network management.

2.5 *Intrusion Detection in Unix*

There are several ways to detect an intrusion in Unix. These techniques do not guarantee the detection of an intruder. However, this is a proactive way to monitor your system(s) to detect common hacker practices. The following technique is provided by [MIS 2002].

2.5.1 Monitor Log Files

Periodically examine log files for connections from unusual locations or for other unusual activity. It is best to automate this as much as possible so that it does not become an overwhelming task or is forgotten. Some tools that can be deployed for this purpose:

- Log Scanner
- Logcheck
- Swatch
- Log Surfer

If more than one system is monitored, send all syslogs to a secured-as-possible loghost (meaning preferably no user accounts and only essential services running). Some of the techniques used for monitoring log files are:

- Check the system logs in */var/log* and */var/adm*. This log files will show what processes have been running and any errors which may occur with them. Examine these logs to determine if there have been any problems with running services or any connections from unusual places.
- Check web server logs for abnormal activity. Review web server log files (if there is a web server running) to determine if there have been any errors or compromises. There are commonly known exploits in certain publicly available cgi-scripts and in cgi-scripts packaged with certain web servers. Review CERT at www.cert.org for these exploits and watch the web server logs for attempted accesses of these cgi-scripts. (Examples include: */cgi-bin/handler*, */cgi-bin/phf*, and */cgi-bin/test-cgi*)
- Check for suspicious logins. A record of user logins is available through the use of the last command. Periodically check the results of this command for suspicious connections from unknown or untrusted sites. Syslogs can also be monitored for logins and logouts from unknown or untrusted sites using some of the automated log checking scripts mentioned earlier.
- Check *xferlog* for suspicious file transfers. If there is a functioning ftp daemon on the system, then a log of all successful file transfers to the system is contained in */var/log/xferlog*. Periodically examine this file and search for suspicious activity. (Sometimes filenames of specific exploits or hacker tools can be noticed.)

2.5.2 Use Commands that Run at a Specified Times to Your Benefit

Unix has a utility called 'cron' that runs commands at specified times. Use this to monitor network interfaces and search for signs of an intruder. Examples of this are:

- Monitor for network interfaces in promiscuous mode. A network interface in promiscuous mode is usually an indicator of a network monitoring program, commonly called a sniffer. Intruders use sniffers to capture username/password information and are commonly included in hacker toolkits. Under some Unixes (SunOS), *ifstatus* can be used to monitor for this. *Ifstatus* is a utility which only produces output if a network interface is in promiscuous mode. This makes it the perfect tool to run frequently as a cronjob. If *ifstatus* will not work with certain versions of Unix, look for other tools which can detect a network interface in promiscuous mode or programs that have the network interface open.
- Look for core files. Some exploits try to overflow or crash programs in order to give root access. When programs crash an image of the running binary is saved as a *core* file to be used for debugging purposes. The file system can be examined for these *core* files on a regular basis. When a *core* file is found, it can be evaluated with the *file* or *strings* commands to determine what program produced the core file. In some instances core files of hacker tools or exploit binaries may be found.
- Search for users that have unauthorized root shells. Use the *ProcTreeNode* custom utility to periodically search the process listing for users which have unauthorized root shells.

2.5.3 Examine the Filesystem

Periodically examine the file system for hidden directories, modified or added files, etc. For example:

- Search for SETUID ROOT scripts. Intruders often leave setuid scripts to easily give them root access at a later time. Monitor the file system for unusual binaries or scripts with the setuid or setgid bits set.
- Search for group and world writable home directories, .forwards and .rhosts files. Allowing accounts to have group and world writable home directories, .forward and .rhosts files opens the system to potential security problems. Periodically search for these types of problems and evaluate if they are necessary. Perl scripts can be written to search for each of these things. In addition to printing the file permissions it prints the contents of the .forward and .rhost files.
- Examine the system for hidden directories. A common technique for hackers on Unix systems is to hide their tools in hidden directories such as '...', '.. ', '.xx' and '.mail'. Use the *find* program to look for these types of hidden files and/or directories. Use the *fileaudit* utility to look for admin defined goofy directories.
- Examine critical files. Periodically browse through critical system files that start/stop services or manage user information/accounts.
- Check for changes in /etc/hosts.equiv and user .rhosts files. These files identify *trusted* accounts and machines, which enable users to login without using a password. In particular Look for '+' entries and inappropriate non-local host names contained in these files. In addition, verify that these files are not world or group writable. The previously mentioned Perl script can be used to monitor the contents of .rhosts files.
- Check for changes to the password file. Periodically examine the password file for accounts with no passwords, accounts with changed shells or uid, and added accounts. It is also appropriate to check for and delete expired accounts.

- Check for unauthorized services. Inspect */etc/inetd.conf* for unauthorized additions or changes. This file contains the list of servers that *inetd* invokes when it receives an Internet request over a socket (i.e. telnet, ftp, etc). In particular, search for and check all entries that execute a shell program (for example, */bin/sh* or */bin/tcsh*).

2.6 Summary

This background chapter provided a literature review on Unix-based intrusion detection. It began with a discussion of computer security in general and proceeded to elaborate on the Unix operating system. It then discussed the application of security in Unix and made a comparison between Unix and Windows NT in seven key operating system security areas. This was followed by an elaboration on the topic of intrusion detection. Finally, the subject of intrusion detection in Unix was addressed.