INTRUSION DETECTION SYSTEMS: AN INTRODUCTION TO THE DETECTION AND PREVENTION OF COMPUTER ABUSE

A THESIS SUBMITTED TO THE
CENTRE FOR COMPUTER SECURITY RESEARCH
DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF WOLLONGONG
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
BACHELOR OF COMPUTER SCIENCE (HONORS)

By
Justin Jay Lister
January 1995

Dedicated to the memory of
Stephen C. Watts

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Bachelor of Computer Science (Honors).

Professor Jennifer Seberry
(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Bachelor of Computer Science (Honors).

Dr Reihaneh Safavi–Naini
(Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a dissertation for the degree of Bachelor of Computer Science (Honors).

Associate Professor Josef Pieprzyk

Abstract

This thesis was partially written to serve as a useful reference aid. It begins with an introduction to computer security. Identifying the confidentiality, integrity and availability issues of information security. We also examine many of problems and vulnerabilities. Some statistics of intrusions is presented to show that there is still need for more effective security mechanisms.

The second chapter looks at the work done so far in intrusion detection, a historical overview of systems is given, from the initial idea of detecting intruders by simple algorithms that analyze audit data, to some of the more advanced systems that have been developed over recent years, In the third chapter, we look at the common components that make up intrusion detection systems.

The fourth chapter investigates some of the problems identified with these systems and suggests some possible extensions and solutions. These ideas are intended to be the focus of my continuing research in the area.

While space restrictions forced me to remove the following chapters, they are intended to be released as reference material (once a few more areas are expanded) for those entering the field of intrusion detection.

In chapter five, we look at some of the commonly used tools that are available to increase password and network security. As well as a few tools that can assist in system monitoring and system checking.

Finally, the appendix provides some details of the response groups that offer assistance in regards to computer security issues. Subscription details for the Intrusion Detection System Mailing List and other related security lists are given for those wishing to find more information about the ongoing work in intrusion detection and computer security. Release information for commercial intrusion detection systems is presented for those looking to use intrusion detection systems. Some issues related to the legality of monitoring for intruders and attempts to track them down is given. And lastly, for those wanting material a list of sources for material is given.

Preface

The inherent lack of security in some of todays popular computer systems has recently become a major issue, evident in the increasing number of system intrusions. Because of our increasing reliance on the computer system, we are confronted with the task of ensuring that the computer systems are used by only those authorized to do so, and that the usage is according to policy. However, there are many increasingly difficult problems in ensuring proper usage of the computer facilities.

- many of todays systems weren't initially designed with security in mind.
- the increasing complexity of the systems, and the many vulnerabilities.
- the complex task of maintaining these systems.
- the lack of experienced administrators and an abundance of inexperienced users.
- inadequate resources for applying security techniques.
- increasing growth in decentralized networked systems.

So, the increasing reliance on our systems has brought a realization of the need for increased security of these systems. As no single mechanism provides us with absolute security, we must rely on multiple, tiered mechanisms. Intrusion detection systems are just one of the many mechanisms that can be used to increase the security of the computer system, ensuring that only authorized users have access to the system, and that all access is according to policy specification.

Intrusion Detection Systems (IDS) is a relatively new area of research. It draws on knowledge from many different fields; Auditing, Artificial Intelligence , Data Analysis, Cryptography, Computation Complexity, Software Engineering, System and Network Management.

The major requirements for intrusion detection systems is the ability to detect intruders or abusive actions in a computer system, in an efficient and timely fashion.

In preparation of this document, it was important to specify a set of goals of what should be expected as a result of the work. The objectives of the thesis were as follows;

- to obtain detailed knowledge of the work done in the field of intrusion detection and computer system security.
- should be able to show an understanding of the work and make intelligent commentary and suggestions towards the strengths and weaknesses of work in the field.

- to have shown initiative and contribute in some meaningful way to the field.

The work done towards achieving these goals includes:

- vast amount of background reading in hacker cases, computer security incidents and unix system services, administration and security.
- preparation of an extensive bibliography of intrusion detection and associated computer security materials.
- collection and testing of many security related tools.
- examining many of the known vulnerabilities.
- initiating and maintaining of the Intrusion Detection System Mailing List, FTP archives and WWW page.
- expression of many ideas that received praise in community:
  - Automated Policy Formalization and Enforcement
  - Universal Intrusion Detection Systems
  - Signature Distribution
  - IDS specific dynamic audit subsystems
  - identification of many problems with current systems, and possible extensions for work already done.
- presentation at 14th Intrusion Detection Workshop (Baltimore MD USA).

Acknowledgements

There are many people I would like to acknowledge for assistance and support during the development of this work.

Firstly, my parents who have always supported me in my efforts (at great personal costs), and always managed to be understanding, a feat which at times, has been no less than extraordinary.

Special thanks to my supervisors Professor Jennifer Seberry and Dr Reihaneh Safavi–Naini, who took on a lot more than they had probably realized. Somehow they managed to keep me focused (well, as focussed as could be expected) on the work that I was supposed to be doing. They have always managed to find the time and take that extra bit of effort, and their willingness to listen, and offer advice has been priceless.

I would also like to thank the other members of the crypt group, especially Associate Professor Josef Pieprzyk, Dr Yuliang Zheng and Dr Thomas Hardjono who have always been keen to listen to what I have to say.

Due to his efforts, also in the area of intrusion detection. Mansour Esmaili has also been of great assistance. The many discussions, sharing of collected material, and ensuring that there was always some paper left over in the printer.

I would also like to thank Kirk Barrett, for his assistance in the enabling the intrusion detection system mailing list. For the lend of the Answerbook (BSM, ASET) documentation and the interesting conversations.

I would also like to thank the NIDES researchers at SRI. Especially, Teresa Lunt who has been a valuable source of papers, and ideas. Debra Anderson for her assistance and efforts with the NIDES Beta and the 14th Intrusion Detection Workshop. And Peter Neumann, for his comments and experience, and especially for not dragging me over hot coals for my seminar at the workshop.

From Haystack Laboratories, I would like to thank Steve Smaha for his contributions of reference materials. Additionally, for the demonstration of his Stalker software, and valuable advice.

Special thanks go to those who made my first conference, and first visit to the US and enjoyable one. Sandeep Kumar from Purdue University, Jeremy Frank, Nick Puketza, Todd Heberlein, Karl Levitt and Matt Bishop from UC Davis, Bob Palasek from LLNL as well as Rebecca Bace from the NSA.

I would also like to thank all the participates of the workshop, who listened and offered comments on my seminar. As well as giving an interesting insight to the developments in the area of intrusion detection.

And finally the members of the IDS Mailing List, for the valuable discussion and contributions.

Contents

List of Tables

List of Figures

Chapter 1

Role of Computer Security

"The only system which is truly secure is one which is switched off and unplugged, locked in a titanium lined safe, buried in a concrete bunker, and is surrounded by nerve gas and very highly paid armed guards. Even then, I wouldn't stake my life on it." - Gene Spafford

"Computer Security is a waste of time." - Some Top Computer Scientists.

The above quotes contrast the two extremes in opinions towards the need for computer security. While the first quote was probably a sarcastic comment, there are some draconian system administrators, who really believe that security can only be achieved if everything on the system is monitored and the users severely restricted.

At the other extreme, we have those who think security is an utter waste of time. It is just a waste of valuable system resources, only makes using the computers more difficult, and that the threat is not large enough to warrant the restrictions. The large number of varied attacks being performed against computer systems (see CERT Advisories Table1.2 clearly demonstrates that this is not true. Users need to be educated about the real threat, and that security isn't a hindrance in–place just to make their life difficult (or justify the administrators job) but is of vital importance in ensuring proper working of the system. Moreover no amount of security will be effective if the users are not aware of good security practice.

The required level of security depends on; What needs to be protected, what threats are considered and the amount of time, money and effort that can be expended.

This chapter begins by investigating the different areas of information security, and the different areas that need to be addressed for effective computer security. Some of the common causes of security problems is also given. This is followed by identifying the threats to our systems, and some of the vulnerabilities that have been exploited in the past. In addition, an overview of some of the more interesting cases that have appeared in press is presented. Finally some of the techniques that can be used to increase the security of the computer system will be given.

1.1 Information Security

Computer Security is a step towards achieving the larger goal of Information Security. A computer system is a means by which we store, retrieve and manipulate data. It is the data that is of principal value; this is not to say that computer equipment are not important, but just that the information it contains is more so. The three commandments of information security are (CIA):

1. *Confidentiality* - is ensuring secrecy of the data. That is, only those who are authorized to see the data can do. Cryptography usually plays an important part in ensuring confidentiality of data.

2. *Integrity* - is ensuring correctness of the data (and programs). Data by it's very nature is vulnerable to modification, and so its integrity relies on proper mechanism to ensure that only authorized entities can create or modify the data.

3. *Availability* - is ensuring data (and service) is available. That is, it should not be possible to prevent authorized users from accessing the data (denial of service).

Security of a computer system includes all the above properties. An interesting analog often used for computer security is medievil castle security. Security of castles rely on a tiered approach (multiple layers of security), so that if one layer should be subverted there are other layers left for protection. The most important objects are kept in the central keep, which is guarded by troops, which are protected further by multiple fortifications (guarded, steep walls, multi–gate entrances, bridged moats, and situated on top of hills). Unfortunately, like many castles, computer systems are also often infiltrated. While their main defenses may appear to be solid[1] breaking the defenses doesn't always rely on breaking the solid defenses.

[1] It is historically important to note, that one should not get a false sense of security after applying a few strong protection schemes. For many castles were defeated due to overlooking the cunning of the enemy (such as using an escape tunnel as an attack entrance).

1.2 Effective Computer Security

To ensure effective computer security, the following areas must be addressed.

1.2.1 Physical Security

Physical security is ensuring that only authorized physical access to computer equipment is possible. Apart from the concern for property theft and damage, physical access to the computer system allows the intruder to shut down the system and possibly reboot such that he/she obtains system operator privileges. Backup media should also be physically protected. The strongest access controls on the computer system are worthless if the backup media is easily accessible. Additionally, systems running mail and file servers, as well as network hardware need to be physically protected, access to such equipment means that the user can possibly reconfigure the entire computer system services, or at least deny others access.

1.2.2 Software Security

Software security is ensuring correct installation and management of software and ensuring its integrity. This can be a very difficult task due the large variety of software packages available for the popular systems. It is hazardous to trust software packages installation scripts, if it isn't trusted software. Such scripts need to be checked (ideally all software install scripts should be checked), as it is simple to hide a process to install a backdoor in a large installation script. Trusting the default settings of software has been shown to be disastrous, with many vulnerabilities attributed to software shipped with incorrect default settings. Also many systems are shipped with large amounts of utility software. Care should be taken to install only software that are specifically required. This is due to the fact that less popular software often contains more bugs due to limited field testing and hence less likely for it be maintained with bug fixes. Large and complex software should have minimal access capabilities as such software has more chances of bugs. But this is often not possible due to the nature of the software. In such cases it is vital to be familiar with the software and ensure bug fixes are applied. Regular backups of the systems files is crucial to proper management of the system software, it ensures that in the event of system corruption (both accidental and intentional) the system can be restored from latest backups. Specific techniques for ensuring software security will be covered later in section 1.7.

1.2.3 Access Controls

Access control is ensuring that only authorized accesses to system services and resources are allowed. Access controls can incorporate both physical and software controls. This includes authenticating the identity of the system users and preventing them from performing unauthorized tasks and/or obtaining access to the system. There are various methods for authenticating users to the system. Firstly, it can use something *the user knows*, in which the user must be able to produce a secret piece of information that only the user and system know. This may be in the form of a secret password, pin number, or the user being able to calculate a response to the systems challenge (challenge–response). Secondly it may rely on something *the user has*, such as a smart card or a key. Finally, it may rely on something *unique to the user*. Common biometric methods include using the users fingerprint, voice patterns, or retina scanning to verify the users identity. Also, another popular method that has been widely explored in relation to intrusion detection is keystroke and mouse dynamics. Where information about the pattern of user typing or movement of the mouse in a graphical environment is used for identification purposes. Also combinations of all three can be used, depending on the level of security required (as well as the available funds).

1.2.4 Policy Controls

Policy controls involves setting and maintaining operating procedure guidelines. Policy documents can be very different depending on the task the computer system is used for. In an Internet environment a policy guideline includes operating procedures of the computer facilities, defines reasonable use of the facilities, and standard etiquette required for services such as mail, news and talk. It should also outline how to set and maintain users passwords including how to choose good passwords, how often to change them and various problems such as the risk in using local passwords on external systems. The policy guideline also outlines penalties for violating the terms of the policy. Usually the users are required to agree to the set of terms (for authorized system use) outlined in the policy document and sign the policy document as legal agreement. This acts as a deterrent towards abusive behavior. Essentially a policy guideline is useful in educating the users to standard operating practice, various resources available to the user and security issues related to ensuring the proper working order.

Enforcing the policy can be a difficult and a resource intensive task. But it is crucial to have one, and that it be as clear and concise as possible. Additionally educating users in security related issues and the importance of security is vital, as users often see security as a hindrance rather than a necessity. User education can often be more effective than a host of complicated security mechanisms.

Methods for addressing the security issues will be outlined further in section 1.7.

1.3 The Common Causes of Security Problems

"It (UNIX) was not designed from the start to be secure. It was designed with the necessary characteristics to make security serviceable." - Dennis M. Ritchie

UNIX wasn't designed with security in mind. Rather it has been an afterthought, a feature that has been slowly integrated over time, often too slowly. While there are various operating systems that have been built around strong security models, they are less commonly used, and sometimes are not practical alternatives.

A brief history of the development of UNIX can be found in [52] (pages 6-8) and [158] (pages 25-30). An outline of security models for operating systems can be found in [121, 137], and [52, 34, 157, 68, 35, 131, 130, 153, 110, 65, 113, 83, 18] are a few good starting places for UNIX security.

References[89] and [87] offer interesting insight to the early days of multiuser system development at MIT (which corresponded to the Multics development as well as John MaCarthy's AI - LISP developments), where the members of the Tech Model Railway Club got up to more than just playing with train switching systems. It quickly became apparent that in these multi–user systems some form of protection or separation of users tasks was necessary. They had developed a game in which players tried to crash the system using the simplest and fastest code.

It is unfortunate that many of the problems recognized in the early days of distributed multi–user systems development are still around today.

Today, we have large networks of computer systems which increase our ability to communicate information all over the world in a matter of seconds. However this comes with an increased risk to security. As we connect our systems up to networks such as the Internet, we will be able to access systems all over the world, however the millions of users on the network will also have the ability to try to access our systems. Looking at the number of alerts or the statistics of unauthorized access attempts[23, 33] (doorknob twisting) combined with the exponential Internet growth rates, it is likely that the problems will only become larger.

Some of the more common security problems are:

- The general problem of networking and increased connectivity equals to increased security risks.
  As explained above, by connecting to large networks such as the Internet the level of exposure to the system will be increased.

- Security is low priority issue.
  In the increased rush towards networking of computer systems, security has often been a low priority. Often there is little or no risk assessment of connecting our systems, or adding new software packages to our systems that are exposed to the rest of the world. Network administrators, and system administrators often lack the resources to concentrate on security issues as in large environments maintaining the operating order can quickly consume all their time. Security evaluation is scarcely of interest when there is increasing demands by users for the latest and greatest software and hardware.

- The chain is only as strong as its weakest link.
  Defeating the security of the system does not require the most secure mechanisms to be bypassed or comprimised. Often vulnerabilities exist due to the security aspect being overlooked. For example, in a network of computer systems there is sometimes a lack of concern for the security in the users workstations, with the administrators concentrating on the security of the larger multi–user systems. To allow the users fast and easy access to the fileservers, the workstations are setup to be trusted as it is assumed that only authorized users are using them. This creates a weak link in the chain, in which the security of the workstation may be easily comprismied by the intruder, hence allowing the main system to be comprimised requiring less effort than overcoming the strong security mechanisms (due to the trusted status of the comprimised workstation).

- False sense of security.
  Even when security is a concern, some administrators have been caught with a false sense of security in their systems. Such administrators, believed their systems were secure because they had implemented strong access control mechanisms, with devices such as firewalls. But unfortunately forgetting to ensure that modem annexes are placed outside the firewall, or that users did not have auto-answering modems directly connected to their machines on the internal network. An ideal analog is to build a 2 inch thick hardwood front door and putting bars on the windows, yet leaving the old screen mesh door out back.

- Security through obscurity.
  One of the biggest problems in computer security is that often people try to obtain secrecy by hiding details of systems. Although this can delay breaking of systems cannont provide any assurance about security. It is conceivable that these hidden details will be eventually found (by reverse engineering of a captured unit) in which case the security of the system can be totally comprimised.

- Inexperienced administrators.
  Because of the technical complexities of the systems and security issues surrounding them, the administrator needs to be widely experienced in all aspects of the system services and security techniques. Finding experienced administrators can be a difficult task and keeping up to date with the latest security issues can be a very time consuming exercise.

- Unfamiliarity with system services.
  Often administrators are not familiar with all the services available on the system. Such services may come pre–packaged so that they run straight out of the box. However, as outlined in the previous section, this can be a open invitation for exploitation. If the administrator is not familiar with the services, then they should not be running. If it is a valued service, then the administrator needs to familiarize themselves with the service and any associated security issues.

- Default settings.
  It is common to rely on the default settings of a program or service. This can be a dangerous assumption as shipped settings have often been the contributing factor in comprimising security. In

addition programs that install software, may specify particular default settings for certain system configurations, and it may be the case that the configuration expected isn't identical as required for the defaults, so it is safer to verify all settings.

- Source code availability.
Another large problem with regards to security is that once a system bug or vulnerability is found, due to lack of source code rights[2] it often can not be immediately fixed. It is even worse when a bug is found in a program that is crucial to the system. So often temporary patches, or work around fixes are used, which may introduce more problems. It is therefore a important to consider software that have freely available source (or software in which source code rights can be obtained cheaply).

- Host hopping.
Host hopping or connection laundering is another problem inherent to large networks. An intruder will most likely (actually more of a certainty due to the trivial nature of doing so) connect through multiple systems before attempting to launch an attack on the intended target, which makes it nearly impossible to track back to the intruder (especially when crossing international borders). This is due to the difficulty in contacting all the necessary administrators, and authorities due to differing timezones. Also attempting to get all the legal permissions can be almost useless when a user can perform their goal and be out in a matter of seconds.

- Breeding rabbits.
Population explosion of unauthorized account users starts out with one person who doesn't think that there is much harm in sharing their account password with a close friend who doesn't have access to Internet. Their friend however decides to break some accounts, so runs a program such as *crack* on local password file to obtain accounts for all his friends to use, who also have friends that would like access ... ad–infinitum. Even worse is when a site gets a reputation of having lax security, activity of crackers explodes as more and more account passwords are shared. This is also a problem as it contributes to the host hopping problem, unauthorized users may trade lists of account/password pairs so they can launder their connections, and using these new accounts they break into new systems, in which they then trade for more accounts. The net effect is, more and more users increasing the web of penetrated systems.

- Lag in legal system.
An important part of computer security is having deterrents towards the crime (unauthorized access of computer facilities). While the legal profession has been slow to adopt new laws in regards to the computer crime, due to lack of understanding and the overall complexity of the technical issues involved. Neither courts, juries or police necessarily understand computers. There is also lack of tangible evidence, such that there are no fingerprints or physical clues. Many issues need to be resolved: What trust can be placed on electronic evidence of logs that can be easily

[2] It is not uncommon to hear about bugs that have been known to exist for years (eg. rdist) before being fixed by vendors, this is one of the major problems as it involves getting others to fix the problem, when you may have little influence.

modified ? How does one put a cost on magnetic signals or what is the value of computer time, especially if the system would have been idle anyway ?

Although there are laws specifying that unauthorized use of computer facilities or stealing of copyright data is punishable by imprisonment but it still continues to exist mainly because it is still very rare to be convicted of a computer crime. This is usually due to the lack of evidence and the technical difficulty involved in prosecuting the case. It is also common to hear the defendants claims of only doing it to show the security flaws of the system (claiming as being out only to help increase the level of security), or that there was no real damage (nobody was hurt so how bad could it be).Often because many crimes involve juveniles, that it was just a childish prank. Also the victims are often hesitant about pursuing legal action because of the monetary and reputation costs.

- Security mechanisms v's System useability.
  Security or the requirements involved with applying security are often viewed as a hindrance, by both users and administrators. Evident by the fact that a lot of systems that have C2 capabilities are not being run at the full C2 level. Maintaining security is a resource intensive process, requiring large amounts of experienced administrators time. On the other hand, neglecting security issues can be even more costly and time consuming.

1.4 Where, Who and What are the Threats

Threats are the potential for harm to our computer systems. This damage may be accidental or deliberate, with the later likely to cause more damage. Making a quick recovery difficult as it is targeted to cause the greatest amount of damage. To prevent such occurences a systematic security plan should be designed and risk analysis performed to determine the areas of maximum risk. So that an approximate assessment of the cost effectiveness of implementing the necessary precautions. More detailed information can be obtained from [121] Chapter 13.

1.4.1 Where are the threats coming from ?

Threats can be categorized as either **internal** or **external**. That is, threats coming from hosts internal or external to our network. The source of threats from users within our system can be classified as follows;

1. *Masqueraders* - are users who are working under the guise of another user, such as using the account of another user.

2. *Misfeasors* - are users who are authorized to use the computer system, but attempt to abuse their privileges.

3. *Clandestine* - users evade the system monitoring and/or auditing facilities.

1.4.2 Who is the threat ?

Some of the commonly used terms that describe the sources of threat are:

- *Hacker* - someone who is considered a computer expert or guru, interested in exploring the depths of computer systems, with no malicious or destructive intentions.

- *Cracker* - someone who is interested in breaking into as many systems as possible and may cause damage attempting to hide tracks or if discovered.

- *Crasher* - someone who intentionally attempts to crash systems, or perform denial of service attacks. No hesitation towards corrupting system data.

- *Phreaker* - someone who is interested in exploring the inner–workings of the phone systems. This may include attempting to obtain access numbers for obtaining free phone calls.

- *Pirates* - someone who is deals in warez, distributing or trading copyright software.

- *Virus* - software program that attaches itself to other programs in order to propagate itself.

- *Worms* - similar to viruses, except that the software program does not rely on a host program to propagate. A worm propagates by copying itself over computer networks.

- *Trojans* - programs that are planted into the computer system under to guise of another program. When invoked the trojan appears to work as the normal would except has that it performs additional operations.
- *Logic Bombs* - programs with embedded code that will cause the program to fail to work after some specified event (such as time or date has passed).

1.4.3 What are the threats ?

While any asset (hardware, software, data, people, documentation and supplies) in the computer facility are possible targets for threats, this section will only deal with some of the possible threats within the computer system. Some of these threats include;

Accounts

User accounts are the first target for obtaining access to the computer facilities. Some of the potential risks of unauthorized users obtaining user accounts are:

- time and money - costly on multiple fronts, if the account is on a pay basis, then unauthorized use is going to be charged to the authorized user, which may result in disputes over account charging. When such a situation occurs, time has to be spent tracking down how the user entered the system. Once the unauthorized users are found and removed, time will be spent ensuring the integrity of the system. So, money is lost paying for the time of the administrator, in addition to the lost charges that no longer can be accredited to the authorized user.
- use and abuse of privileges - when an unauthorized user obtains access to another account, they obtain the privileges of that user. So if the user account had special privileges or access to valuable data, then the unauthorized user could then possibly disclose, delete or modify the data.
- annoy other users - user accounts can also be used to harass other users, by writing to the users terminal, sending talk requests, or sending abusive mail or news posts.
- hop stop - the user account may simply be used as a temporary access point to break into other systems. This might include using the account for temporary file storage.

Data

A large threat to any system is the ability for potential data copying, deletion or modification. This includes activities such as:

- reading private mail.
- copying assignments or modifying results.
- distributing trade secrets or proprietary software.
- corrupting filesystems.

Denial of Services

Denying others access to computer resources.

- hogging CPU, disk or network resources.
- crashing machines.

1.5 Reported Incidents

Many hacker incidents[3] have been reported in the computer and general communities in recent years. There has been no lack of enthusiasm in the press to report such incidents (many have been exaggerated or unsubstantiated, the CERT advisories identify two incidents reported in the press that were never substantiated). It is actually quite remarkable that there were so few, considering the amount of interest these stories have received in the past. The cult movie Wargames has been attributed to introducing more young teens to the world of computer and phone hacking than any other. Many other films such as Sneakers, Drop Zone and TV series such SeaQuest DSV, Star Trek the Next Generation (STNG), Bablyon 5 and Cobra have also focused on the art of hacking.

The following section will present an brief outline of some of the more notable incidents. Furthermore, each section also includes references which make for very entertaining reading.

1.5.1 Kevin Mitnick

The book *Cyberpunk* Part One[59] describes the story of Kevin Mitnick, one of the most active (and knowledgeable) phreaker–crackers of recent years.

Mitnick started out in the phreaker scene back at the beginning of the 80's. His other cohorts during the 80's included Roscoe, Susan Thunder, Steven Rhoades, Mark Ross and Lenny DiCicco.

Mitnick was obsessed[4] with obtaining unauthorized access to high profile computer systems. He obtained unauthorized access to many educational computer systems (high–school, college, and universities) in particular the University of Southern California (USC) and California State University (CSU) which were often used in many of his exploits (mainly for file storage and network access).

MCI, Pacific Bell, TRW, SCO, DEC were among the Commercial computer systems compromised by Mitnick. Even military computer systems were not insusceptible to his efforts. He obtained access to the National Computer Security Center's (NCSC) dockmaster computer system, which is used by military contractors and users from the National Security Agency (NSA). Also a Department of Defense (DoD) computer system Elex-Wash was penetrated.

Mitnick held vast technical knowledge of the phone and computer systems (often more technical than the operators themselves) as well as an uncanny knack for obtaining any information he required through social engineering techniques.

Some of his early phreaking pranks included acts such as; redirecting telephone assistance lines in Rhode Island. In which, automated messages like: "Is the person white or black, sir?" .. "You see we have separate directories." Or "Yes the number is 8750 & $\frac{1}{2}$" ... "You do know how to dial a $\frac{1}{2}$ dont you maam ?" were installed.

Mitnick was able to access computer systems all over the US for free, using MCI account codes he easily obtained from MCI's network system. By assuming control over large portions of Pacific Bell's Los

[3] It is of interest to point out that this doesn't agree to the definition of hackers given previously. It was unfortunately the buzz phrase picked up and used incorrectly by many reporters, and now it has seemed to have stuck - to the regret of many true system hackers.

[4] In the trial for obtaining unauthorized access to DEC and copying the VMS 5.0 operating system, Mitnick was sentenced to 6 months in a community program to deal with his hacker obsession in addition to a 1 year prison sentence.

Angeles telecommunication computer systems. Kevin was able to initiate or monitor any phone trace attempts, deactivate or activate services, forward calls (or traces) to random numbers. Unauthorized access to the computer systems at TRW allowed him to perform credit checks, in addition to being able to modify others credentials.

Checks for pending warrants from the LAPD were often performed. In one incident he had found out about a warrant the morning it had been issued. Mitnick avoided the warrent when he fled for a year returning a few weeks after the warrant had expired.

He attempted to obtain by illegal access to SCO's a copy of their XENIX operating system, also attempted to obtain a copy of VMS from Pierce College. Mitnick was later successful in downloading DEC's entire VMS V5.0 operating, from DEC's own software development cluster. Utilizing a gateway from DEC's EasyNet to Internet (via gatekeeper.dec.com) to store the copy on a host at USC.

While obtaining the VMS code, he had defeated DEC's top security team attempts to trace the him. By monitoring their email, he obtained a copy of the Chaos Computer Club's trojan login software (loginout). Also a copy of a security tool (XSafe) was obtained from one of the security teams own personal account.

1.5.2 The Internet Worm

Robert Taipan Morris (rtm), the son of esteemed computer security expert (who worked for the NCSC-NSA), was responsible for the creating the Internet Worm program that was released on the evening of 2nd November 1988 and was responsible for infecting an estimated 6000 BSD derived computer systems (largely Sun and VAX).

The program collected host, network and user information, the broke into machines using the following three flaws:

1. sendmail (using debug option) - allowed mail messages to be sent to programs, which would then run the commands in the message. Mostly effective on 4.3 BSD and Sun machines, as they shipped with this option enabled.

2. fingerd - exploiting the finger daemon involved overflowing the input buffer (as the daemon used a gets() function that lacked bounds checking). Since the buffer was maintained on the stack, overflowing the buffer allowed a fragment of code to be executed when the function returned. Only 4.3BSD VAX systems were susceptible to this attack.

3. exploiting remote execution facilities rexec and rsh (trusted hosts). The worm attacked accounts (via rexec) using obvious passwords (permutations of obtained user details), it's own 432 word internal word list, and the system dictionary /usr/dict/words. It also utilized trusted hosts (/.rhosts, /etc/hosts.equiv, ~/.rhosts and ~/.forward) to attempt to propagate to other systems.

To prevent the worm from being discovered it featured a mechanism to control the number of copies allowed on an infected system. However, this section of the code was flawed and the major reason that the worm was noticed as many copies quickly consumed the infected systems resources.

In addition the worm program also featured many self protection features:

- Erased it's argument list once it had finished processing them, so that they wouldn't be discovered in process status command listings (ps).
- Deleted the executable binaries from disk once running. With short running period made it difficult to obtain a copy of the binary.
- Modified resource limit functions used to prevent core dumps, so that no traces were left around due to bugs.
- Named the running process sh (Bourne Shell) as it would not appear conspicuous.
- Constantly changed it's pid, by forking a child process (which continued from the current place) and killed the parent. Which also had the effect of restarting CPU and memory usage.

However it never directly attempted to obtain root privileges (almost never broke in as root), and also at no time did the Worm program try to destroy and data (the only data deleted was it's own).

References [139, 43, 146, 150] give a detailed analysis of the worm program, detailing the specific mechanisms and functions of the worm. In addition an account of how the worm was found, studied, reverse engineered and how patches were developed to "immunize" systems from becoming infecting. Also brief chronology of the spread and eradication of the worm are given. Cyberpunk [59] (Part 3) and [39] also provide additional details into the events surrounding the worm incident, the various opinions towards the development of such a program in addition to background details on the Morris family and the legal implications that followed later.

1.5.3 West German Hackers

Stoll [161, 160, 159] describes the events that lead him to detect the hacker in Lawrence Berkeley Laboratories (LBL) computer systems. From his detailed notes taken during the many months spent tracking down these international intruders, Stoll offers an interesting insight to the activities of the German hacker ring. Markoff and Hafner [59] (Part 2) detail the incident from the German perspective. It offers any interesting insight into the additional activities of the WGH's, how they had meet the East German and Russian agents and made cash deals for obtaining US military data and commercial operating system source code. And the resulting trials of the group.

Other articles (7 & 8) can be found in [39].

1.5.4 The Hacker Crackdown

An account of US authorities *Operation Sundevil* is given in [156]. This may also be obtained from Project Gutenberg EText;

```
The Hacker Crackdown - Law and Disorder on the Electronic Frontier
by Bruce Sterling
January, 1994  [EText #101]
ftp://mrcnext.cso.uiuc.edu/etext/etext94/hack11a.txt
```

The hacker crackdown is an account of the early phreaker years. It outlines a series of early phreaking incidents including:

- 12 page E911 Document which was obtained from BellSouth AIMSX computer network and later published in Phrack. E911 was a technical manual that describe the workings of the 911 emergency phone system, which resulted in many raids and subsequent court cases relating to the distribution of the document. The document was originally valued by BellSouth Security at $79,449. However was later found to be worth less than US$13. Which was the price of publically available documents from a Bellcore catalog that contained significantly more detailed information on the same systems.
- Martin Luther King Day Crash in which AT&T long–distance phone system breaks down and is attributed to phreakers, hacking the telco switching and computer systems. Which was later revealed not to be the case.
- Operation Sundevil, 7-9th May 1990 widespread raids on phreakers. In which 42 computer systems were seized, 25 of which were running Bulletin Board System's (BBS's). Operation related to suspected credit card theft and telephone code abuse. These BBS systems carried a host of information on such topics as carding, programs for scanning telephone codes, information on raiding credit card companies, pirated software, stolen codes, hot credit card numbers, cracked passwords, intrusion manuals, blue–box schematics, anarchy files, etc.

The operations involved the US Secret Service (USSS), the Chicago Computer Fraud and Abuse Task Force, the Arizona Organized Crime and Racketeering Unit as well as telco security personnel from AT&T Corporate Information Security and Bellcore Security.

In addition it offers an interesting overview of the evolution of electronic communications systems, the early developments of the phone systems, the dismantling of AT&T, earlier UNIX, the phone system crashes of 15th Jan 1990, 1st July 1991, and 17th Sept 1991. Also the hacker/phreaker culture that lead to the development of various electronic magazines (2600, Phrack, P/HUN), phreaker and hacker groups (414, Atlanta 3, LoD, FoD, etc). In particular the events leading to the foundation of the Electronic Frontier Foundation (EFF). Also some background information on the founders of Apple, and Lotus Development Corp.

1.5.5 Berferd - AT&T

Account of incidents at AT&T in which an intruder enters the system and is studied, by setting up a host in which the intruder is lured into so that his methods could be studied. Outlined in the book [34], and papers [33, 23].

1.5.6 Internet Monitoring

A personal experience with an actual incident on paxnet.com.au. Had identified the same techniques that had been reported by CERT (CA-94:01.ongoing.network.monitoring.attacks; also see Promiscuous Mode Vulnerabilities in Section 1.6.4) being used to obtain user authentication data. However the attack had not yet been carried out in full, and shell command logs were able to reveal exactly how the user had obtained privileges. The software that was being used to monitor network connections was found and the recovery task performed; changing user accounts, checking system binaries and file permissions, and installing security patches. The full source code for the /dev/nit monitoring software was later obtained.

The above incidents highlight some of the critical weaknesses (many of which have been outlined previously). Many companies and institutions are often slow to respond to reports of vulnerabilities. Furthermore, it has been difficult to get vendors to release details on the known security vulnerabilities and to develop and release the fixes in a timely manner.

The lack of interest in legally pursuing the culprits. This is due to the difficulty in finding substantial evidence to support a case. In addition to the cost of time and effort of personnel and possible loss of reputation as a result of press coverage highlighting the security failures (which can often lead to far greater financial losses than the original incident), so usually it means that they have opted for the easier path of dropping or reduce charges in return for details of how their security failed.

What is even more interesting from the above incidents is the apparent connection between many of these intrusions (especially Mitnick incidents) and the developments of Intrusion Detection Systems. The developments of TRW's (Discovery), DEC's (Polycenter Intrusion Detector), and the NSA NCSC (Midas) systems all occur after the major incidents were reported. Additionally the Internet Worm was the basis for the development of the Computer Emergency Response Team (CERT). There are many opposing views to whether such incidents are beneficial to the development of secure computing systems. This is an area which I do not wish to address, however [147].

1.6 Under Attack - Vulnerabilities

Due to the design of Unix, it is possible to obtain full access (super–user) by exploiting vulnerabilities in the operating system, system services or binaries. Many of the reported incidents are often due to crackers exploiting one or more of these well–known vulnerabilities.

1.6.1 Getting in the front door

Usually two steps in the intrusion scenario. First, access to the system is required. An external cracker will attempt to obtain a user account on the target system. This can be achieved through a myriad of methods:

1. *Social Engineering Techniques* - techniques to fool users into releasing their access details. Such methods as telephone impersonation, where the intruder calls up posing as a user who has forgotten their password or a technician that needs access to service the system and requests that they be given access. Mailing requests to users to modify their password, or giving users of the system trojaned programs. Sometimes going as far as the culprit offering services on their own systems, relying on users who access the service to re–use account details. Which is unfortunately common, as many users find it easier to remember a single password than many different ones.

2. *Guessing or Cracking Account Passwords* - a cracker may try to break in by running automated programs to guess passwords. Cracking passwords relies on first obtaining system password files, which may be a simple task if information services are incorrectly configured.

3. *Dumpster Diving* - collecting information about the users, operating system and any access methods by searching dumpsters. Such information can be useful in mounting the previous attacks. Additionally many network services reveal vital information about the system, that can assist the intruder in mounting an attack.

4. *Remote Services Vulnerabilities* - utilizing insecure remote access facilities. Often services such as ftp, mail, telnet, irc are misconfigured, allowing the intruder access to the system.

1.6.2 Obtaining super–user privileges

Once access to an account on the system (or the ability to interact with the system; to input, execute and retrieve output) has been obtained. The intruder can then proceed with the second phase of the intrusion, to obtain root privileges. This may be via a single user step process ($user1 \rightarrow root$) or multi–user step process ($user1 \rightarrow user2 \rightarrow \ldots \rightarrow root$). An authorized user has the advantage of beginning at the second phase, as they already have access, unless they have only limited access (such as restricted shells). If they have only limited access they would be required to obtain better privileges (which is the second phase problem).

Once root access is achieved they have potentially unlimited access to the system. A likely scenario is then proceed to install trojaned binaries. This can also be used to obtain further account password (for trading, or launching attacks at other systems), or just as backdoors in case the intrusion is discovered and to remove any traces of the intrusion that may appear in the system logs.

Crackers are known to possess lists of vulnerabilities and the tools (programs and scripts) to exploit them. This material, like account passwords is often shared or traded amongst the cracker community.

1.6.3 CERT Advisories

Part of the CERT agenda is to release information (CERT Advisories) about known computer security problems, in addition to working with vendors to provide patches or workarounds.

Table 1.1 shows the number of advisories that have been released by CERT since it began in November 1988.

Year	88	89	90	91	92	93	94
Number Advisories	1	7	12	23	21	19	14

Table 1.1: Number of CERT Advisories since November 1988.

Figure 1.2 shows the number and nature of these advisories.

Nature	# Advisories	Nature	# Advisories	Nature	# Advisories
.files	2	/dev/audio	1	/dev/nit	1
Monitor	1	NFS	3	NIS	2
NetInfo	1	bosperf	1	bash	1
chroot	1	console	1	cron	1
crp	1	decode	1	div zero	1
emacs	1	expreserve	1	file permissions	3
finger	1	fsirand	1	ftp	6
gopher	1	help/printer manager	1	icmp	1
install	1	irc	1	loadmodule	2
login	6	lpd	2	mail	4
majordomo	1	mountd	1	npd	1
packet routing	2	passwd	11	qmgr, migmgr	1
rc files	1	rcp	1	rdist	2
restore	2	rexd	3	rlogin	1
rnews, npasswd	1	rsh	1	sendmail	5
su	1	sudo, smount	1	suid	11
sum	1	SunView	1	tar	1
telnet	5	tftp	3	trusted hosts	2
utmp	1	uucp	1	xterm	1

Table 1.2: Nature of the CERT Advisories.

In addition to CERT advisories, 8lgm also releases advisories with full disclosure[5]. 8lgm released exploit scripts (up till advisory number 8) in an attempt to force vendors to develop patches and so that administrators could determine the exact nature of the vulnerability (unlike CERT advisories in

[5] The argument about what information should be released (ie. full disclosure v's security alerts after patches are available) is a complex one. Many security professionals argue that releasing exploit scripts only invites more incidents. While others argue that CERT and the likes are great black holes, where many vulnerabilities are reported but vendor's will sit until absolutely forced to do something.

which a general description is given only after patches are available). Figure 1.6.3 shows the number of advisories for the various computer systems. The number of CERT advisories are left justified, and the number of 8lgm are shown on the right.

at(1): 1	autoreply(1): 1	binmail(1): 2	gopher(1): 1
login(1): 1	lpr(1): 1	passwd(1): 1	popen(3): 1
prwarn: 1	pt_chmod: 1	rdist(1): 1	sadc(1m): 1
sendmail(8): 1	suid_exec(ksh): 1	tmpfs: 1	urestore(1): 1

Table 1.3: Nature of 8lgm Advisories.

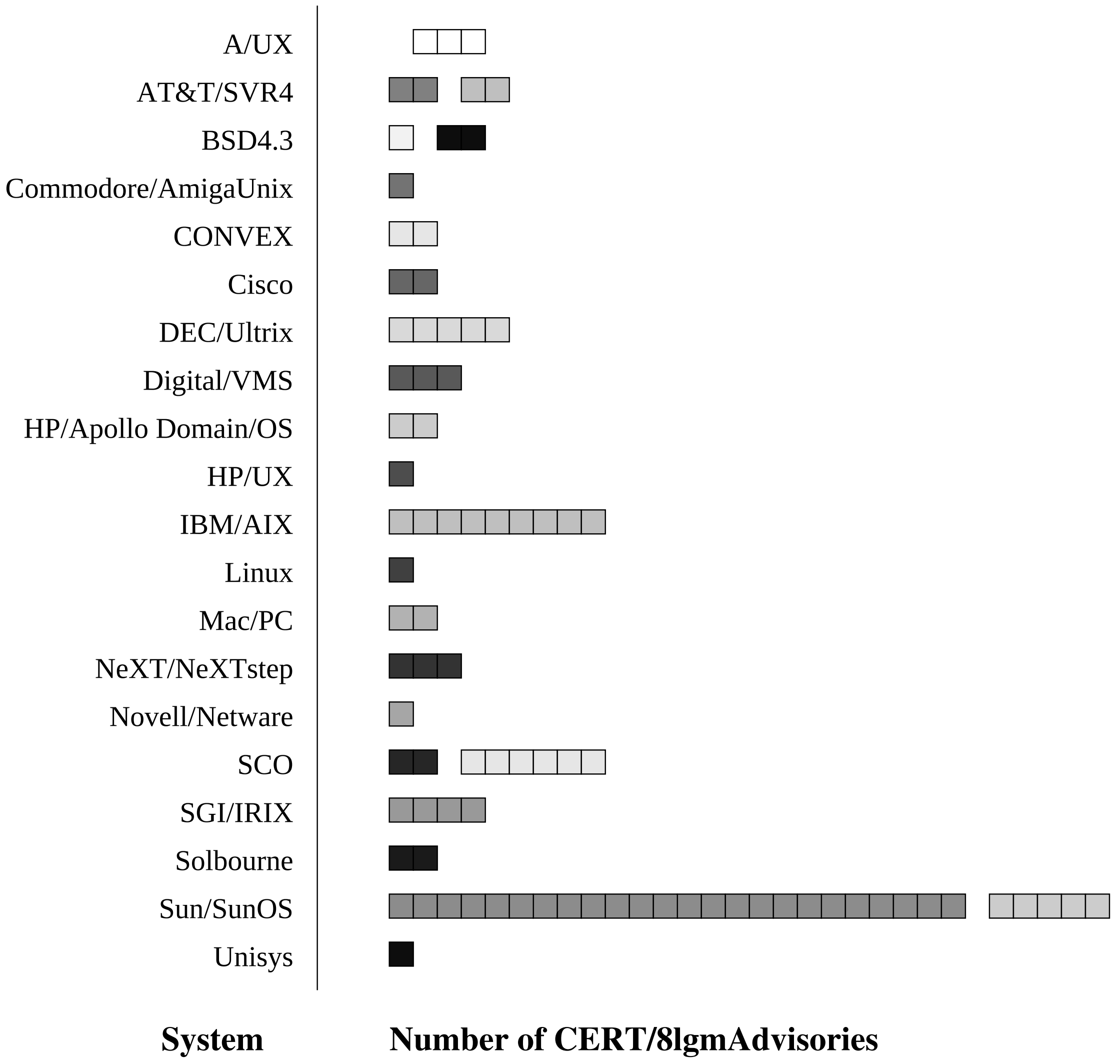

Figure 1.1: Number of CERT/8lgm Advisories for various computer systems.

To get an idea of the number of incidents "In the first nine months of 1994, CERT logged 1,517 incidents - up more than 75% from 1993 - some of them involving networks that link tens of thousands of machines".

1.6.4 Potential Vulnerabilities

Ensuring security of the system has been shown to be a difficult task. It has been shown that there are many vulnerabilities, and have looked at some of the incidents that have occurred. Also the process in which a system is penetrated was detailed.

The following sections investigate some of the specific vulnerabilities that are commonly exploited:

Account Security

The following mechanisms are ofte exploited in various ways, to assist in obtaining access to user accounts.

- *$PATH* - is the environment search path in which the shell looks for commands. Incorrect search paths such as $PATH=.: (a dot as the first path looks in current directory first) can cause the user to execute unintended programs planted by others when working in publically accessible directories such as /tmp. This mechanism can be exploited to allow access to the user accounts (by creating suid programs, or changing users file permissions or contents). In addition the planted program (for example a trojan ls) carries out the normal operations of the program it is faking. In addition to the extra devious tasks, so that the user would be unaware that any extra activity has occurred.
- *umask & $HOME directory permissions* - the user controls access to their files by storing them in their allocated home directory ($HOME). The umask (user file creation mode mask) controls the default file creation mode (actually is a mask of the system default create mode). This directory should be mode 700, which only allows the user access to the files in the directory tree. Or at minimum 755, which allows for other users to search and possibly read files in the users home directory.
- *.dot files* - files in users home directory which are used by programs to configure programs including startup or shutdown scripts. Also they can be used to control access to user account and interface (.rhosts, .Xauthority see below). These files can be exploited to start processes the user never intended. Common dot files are for shells (.profile, .shellrc, .logout), editors (.emacs), network services (.netrc, .tin, .forward, .gopherc, wais, .ircrc, .ztalkrc), GUI (.Xauthority, .xinit, .wmrc).
- *.rhosts file* - commonly used either to obtain access or to ensure that access is available in the future should the real user change the account password. By inserting entries (or exploiting current entries) in .rhosts files access to a system account is possible without entering a password.
- *.forward & .procmailrc files* - user controlled file for specifying processing of received mail, these can be exploited so that specially constructed mail items can run commands specified by a remote user without having to access the account.
- *.Xauthority* - like .rhosts files control access to the account, these programs can be exploited to allow access to the X windowing system. Many systems do not use the xauth mechanism, which in itself is not entirely secure as advanced attacks attempt to capture the *MIT-MAGIC-COOKIE*.

- *batch jobs* (at & cron) - mechanism that allows user jobs to be queued for later execution. Masquerader can possibly hide tasks in user job queues. The user wouldn't realize unless they checked the jobs queue or unless output from the job was not adequately disposed of.
- *suid binaries* - set user id programs run at the effective user id of the owner of the program. This allows programs, such as a shell to be suid to the user and hidden in an accessible directory. Any user invoking the suid shell is then effectively running as the user id of the owner that suid the shell (common backdoors for access to user accounts).
- *screen* - virtual terminal program that multiplexes the physical terminal to allow multiple interactive shells. A feature of the screen program allows detaching and placing in the background, which can be resumed later. It is possible install suspended screen processes to be resumed later, so that a user would be unaware of the running process.
- *.space directories* - dot space directories or directories with unprintable character directory names are also often used by masquerading users, so that files they create are not readily noticed by the account owner. Also writable directories outside the users home directory are often sought to store the masqueraders files to avoid detection from the system administrator and account owner.

Password Security

The most common access control used is the account password, in which the user must know the secret password that corresponds to the account. The account password is used to authorize a user to access to the user account, thus it is the most common target for exploitation. Furthermore, there are many techniques that can be exploited to attempt to obtain account passwords.

- *Fake Login Screens* - users create programs that clone the appearance of the account login prompts. Usually a program is run in another account that displays the system login prompt. When the user enters a account and password pair, the program records the password and reports "Invalid login try again" (or whatever the standard error message is), in which the program then terminates and the real login prompt is displayed. Hence the user is none the wiser. More sophisticate spoofing programs continue to run instead of exiting back to the proper login screen. Acting identical to the standard terminal program, invoking the correct login processes and handling session exits the fake terminal program can be used to log many users account passwords.
- *Keystroke Logging* - keystroke monitoring of computers is also another commonly used technique. Laboratories of PC's terminals are often susceptible to memory resident programs (TSR's such as phantom, PW) that can be used to log keystrokes to a file. In Unix based systems it is also possible to monitor user's /dev/tty's or to monitor X keystroke events.
- *Cracking Password Files* - there are many programs that can be used to crack a system password file (crack, crackerjack, crackerhack, killer crack, etc).

  To attempt to crack passwords the user requires access to a file containing the encrypted passwords (which may be difficult if they are in shadow files as only root processes can access the file), however there are plenty of additional vulnerabilities that can be used to obtain a copy of the password file (such as ypx see below). Because the function used to encrypt passwords is a cryptographic one–way function[6] that was designed to be slow and non–invertible, so that brute force dictionary attacks were difficult. The method of cracking a password is find a word (from a dictionary or list of common passwords) that encrypts to be identical to entries in the password file. Due to the fact that users often use badly chosen passwords, and that these programs use ultrafast crypt function's these dictionary attacks will often crack passwords.

- *Mail Requests* - fooling users into revealing their passwords is probably the simplest way to obtain accounts. Common tricks include mailing users, by spoofing mail as coming from the system administrator in which the user is requested that they change their password temporarily, while some system task is performed. Then is it a matter of searching the accounts to see if any user changed their password as specified.

- *Remote Services* - another trick often employed to fool users into revealing account information. Relies on the fact that many users reuse their passwords for many accounts (so they do not have to remember lots of different passwords). By advertising access to a remote services in which the user's will be interested and requiring the user to enter account details for the remote system it is likely that the password used is the same as the one used for the users home system.

Network Security

Network attacks are becoming increasingly popular. Due to weaknesses in the authentication mechanisms of many protocols, in fact most network data can be is easily modified allows many possible techniques for exploitation.

- *Sniffers* - monitoring packets on the local network is an especially dangerous vulnerability, due to the fact that most passwords travel as cleartext over the network. Also such monitoring is largely undetectable. Again PC's and Macintosh laboratories are a large threat as any machine can be used to monitor the network.

- *Promiscuous Mode* - worse is the number of attacks (see CA-94:01) that have used central Unix machines ethernet's promiscuous mode (/dev/nit) to capture host and user authentication information in newly opened ftp, telnet and rlogin sessions. These attacks required the intruder first obtain root access to the target machine (through some other mechanism), then the monitor program can be installed to capture all network connections authentication data (telnet, rlogin and ftp). Associated with this attack system binaries were often trojaned to further hide the network monitoring process. It has been estimated by CERT that access information for tens of thousands of systems has been compromised via this attack.

[6] Specifically it is 25 iterations of the DES function that uses the password as the key to encrypt 8 null bytes (or 16 depending on implementation). There is also a 2 byte random salt (generated when the password is created and concatenated to the encrypted password data in the password file) value that is used to permute the DES expansion table entries, this allows $2^{12} = 4096$ possible encryptions for the same password (ie. so that identical passwords appear different in password file).

- *Spoofing* - various number of attacks achieved by forging packet header data. Such methods have included faking host addresses, servers, and connection data. These attacks generally bypass the authentication process of the various network service protocols, by exploiting trust of lower numbered (< 1024) ports.

  Spoofed *Internet Control Message Protocol (ICMP)*. Redirect messages can be used to create new paths to a destination.

  Source routing packets can be used to impersonate trusted machines. A TCP connection can be established with forged destination address (that of a trusted host), using source routing to specify the specific path from the real destination to the source (as RFC1122 specifies the destination machine use the inverse path as the return route).

  Bogus *Routing Information Protocol (RIP)* packets can be used to divert traffic due to weak authentication by routers and hosts.

  Also *IP-forwarding* requests can be used to exploit a bug in Portmapper/NFS. Packets sent to a remote NFS server via portmapper can be made to appear as coming from the localhost, in which the NFS server allows the filesystem to be mounted.

- *Denial of Service* - packet storms can be easily created with broadcast pings and the echo port. In which a forged packet can cause hosts on the local ethernet to continue broadcasting back and forth between each other.

Services Security

Misconfiguration of system services has often lead to widespread security vulnerabilities. Some common misconfigurations such as:

- *FTP* - user ftp owned ~/ftp allowed anonymous users to modify the filesystem. Due to copying the system /etc/passwd (which contains encrypted passwords) to the ftp/etc directory when all that it is used from the file is the username to user id mappings for file listing commands. So anonymous users could simply obtain a copy of the system password file.

- *YP/NIS* - /etc/hosts.equiv or /etc/passwords containing "+" entries are used to specify that yellow pages or network information service is running so that server should be contacted to obtain required information. However entries in the server files meant that users could login as user + (resulted in being given root access) without a password.

- *NFS* - world exportable r/w filesystems in which users with root access to local system can create suid programs owned by any non–root (Sun defaults had user bin own /etc thereby allowing /etc/shadow to be modified; note NFS root mounted filesystems mapped to user nobody) user on the mounted filesystem. Which can later be exploited to obtain the access.

1.7 Defending the Castle

"An ounce of prevention is worth a pound of cure." - Russel L. Brand.

This section presents some of the general tools available to assist in increasing the security of the computer system.

There are many freely available tools that can assist in enhancing the security of the computer system. These tools address some of the vulnerabilities that have been outlined earlier. In addition some of general hints are given to avoid some common security problems.

Access Controls

Access controls should be structured so that software runs with minimal access privileges. Often system software requires privileged access to particular system resources and so often run as root (such as sendmail) however it is often the case that it is not necessary that it requires full root privileges and mechanisms exist so that the root privileges can be removed and the software setup with it's own uid and gid. Only critical suid programs should remain (as seen in the CERT statistics suid programs are the largest cause of system vulnerabilities). For NFS systems exporting read/write filesystems, system directories should be owned by root, otherwise remote users can modify key system configuration files, as is possible in Sun systems that ship with bin owned /etc directory.

Some of the available tools commonly used are;

- *sudo* - super–user do allows users to be authorized to execute commands as root. Features logging and additional password authentication for all sudo users activity.
- *chrootuid* - allows commands to be run in restricted environments, allowing network services to run with low privileges and restricted file system access.
- *Osh* - operator shell[114] can be used to distribute specific root level privileges without giving full root access. Osh is a suid root, security enhanced restricted shell that allows fine–grain distribution of system privileges.
- *smrsh* - sendmail restricted shell, limits the environment sendmail operates in. Acts as a replacement of the sh used by sendmail, allowing the commands executed by sendmail to be restricted.

Cryptography

It is possible to employ cryptographic techniques to increase the security of files, mail, talk, irc. Using programs such as PGP (RSA, IDEA and MD5), or standalone cryptosystems such as DES, Loki, or Haval we can increase the secrecy and authenticity files and communications.

For example, PGP maybe used to encrypt (using IDEA session keys, which in turn are encrypted with recipients RSA public keys) and sign (using MD5 and senders RSA secret key to generate the digital signature) mail messages. This allows users to authenticate the senders of mail messages, in addition to ensuring only those intended to read the message are able to. Keyservers have been setup to maintain public key repositories, in which signed public keys can be exhanged/retrieved. Many additional programs and scripts allow PGP to be easily added to standard mail packages (ie. elm, pine), editors (emacs) or ytalk (pgptalk).

Firewalls

Firewalls can be used to control network access between networks, typically one or two systems are used to contruct a firewall. A firewall system essentially acts as a wall between the external (ie. Internet) network and the internal (protected) network, in which traffic can be configured to be blocked or permitted. Allowing a single control point for maintainence of user authorization for services in addition to allowing extensive logging to be maintained.
Packet filters are used to block particular (incoming or outgoing) network traffic (data packets) based on packet header information (ie. src, dest, port). Typically no additional cost for filtering abilities as they come with router software, and a router is commonly required for a Internet connection.
Proxy servers (application–level gateways) installed on the firewall system control the particular services which authorized users can use to access services from either side of the firewall system. Instead of controlling traffic flow, special purpose code for each required application is installed on the firewall (gateway) host, which is used to access services on the other side.

- *drawbridge* - bridging filter for PC with two ethernet cards to perform packet filtering. Part of the TAMU package[133] of tools. Allows filtering based on service protocol, source or destination address and traffic direction.

  ftp://sc.tamu.edu/pub/security/drawbridge

- *screend* - kernel patches for BSD/386 to allow the UNIX kernel to perform packet filtering [108, 1].

  ftp://gatekeeper.dec.com/pub/DEC/screend

- *TIS ftwk* - Trusted Information Systems Firewall Toolkit[2, 3, 4, 127]. Software kit for building and maintaining internetwork firewalls. Included in the toolkit are application proxies for ftp, rlogin, sendmail, and telnet as welll as strong user authentication.

  ftp://ftp.tis.com/pub/firewalls/toolkit

- *socks* - versatile clients, which can run direct connection to inside hosts and proxy connection through SOCKS server to external hosts. Supports dual–homed or multi–homed host configurations.

  Sockified finger, whois, ftp (standard, ncftp, llnlxftp, ftptool), telnet, xgopher, xmosaic, xarchie, irc, ping clients. Also provides details to convert network programs to SOCKS clients.

  There is a devoted mailing list for SOCKS mail; To: majordomo-request@syl.dl.nec.com Body: subscribe socks.

- *xforward* - provides user–level X forwarding service for system providing packet filtering[171].

- *tcpr* - perl scripts to run ftp and telnet across a firewall. (SunOS 4.1)

  ftp://ftp.alantec.com/pub/tcpr

User Authentication

Ensuring only authorized users obtain access to the system is critical. As has been previously mentioned, password mechanisms are commonly exploited by attempting to guess or crack passwords.

Password Software

- *Shadow* - extension to standard /etc/passwd system in which encrypted password data is maintained by root only accessible functions. This ensures only system processes that require the encrypted password data obtain access, restricting users from obtaining password data, to try and crack.
- *passwd+* - proactive password checker, ensures passwords are not of specific (ie. easily guessable) form (ie. dictionary words, permuatation of user data, pattern rules). Additionally performs extensive logging of all password management functions.
- *OPUS* - Observable Password Usage System [149], another proactive password system except that rules are encoded by hashing unusable passwords into hash table bitmap. Bitmap is hash of common guessable passwords.
- *S/Key* - One-Time Password System [60] provides authentication over network that are subject to eavesdropping and replay attacks.

  ftp://crimelab.com/pub/skey

Chapter 2

Intrusion Detection Systems

At some point we expect that the security of the system to fail. Therefore we need to be able to find and report these failures as quickly as possible. Additionally we need to be able to verify that authorized users are working according to requirements (access controls, policies etc). We will see the task of intrusion detection is a difficult one, that require many compromises in terms of practicality and available computation and I/O resources.

In this chapter we look at a few of the projects to develop computer programs that are capable of detecting suspicious or intrusive behavior in computer systems via audit trail analysis. These systems were initially developed due to the difficult and time consuming task of checking system audit records. The volume of audit produced by most systems often means that auditing is disabled or in the situations where auditing is required that either large amounts of human resources was required to scan the data. This lead to the development of programs for automated computer analysis.

Most systems provide a mechanism for auditing (or accounting) the events on the system. Some are better than others, we will see later that many of these are fairly inadequate for use in intrusion detection. Intrusion Detection Systems (IDS) attempt to automate the process of collecting data on the events of activity on the target computer system. From this data (audit trail) IDS then analyze for patterns of activity that are either suspicious, intrusive or anomalous (ie. do not correspond to normal users activity) and alert the system administrator to such activity. Some systems attempt to go further allowing automated electronic countermeasures such as terminating process, jobs or user sessions.

The chapter starts by presenting a background of intrusion detection system research. By presenting a list of the systems we should be able to get a better picture of the focus and effort that has occurred so far. Each system is presented with details on who designed, implemented, and tested the system as well as the environment the system was developed for, including some of the key features of the system. Obviously it is not possible to present every piece of work in vast detail, but by examining a few of these systems it should be possible to present a rough idea of the main developments.

A basic outline of a few of these systems is be presented, along with some of the developers specified goals. This allows us to assess the practicality of the systems design and get an idea of some of there limitations. A small discussion on the researchers results and claims is also provided.

In attempting to provide a discussion on the technical abilities and the practical implications of these systems. Some of the following criteria were used;

- *Processing Capabilities:*
  - whether system is a batch system in which audit is processed daily or weekly.
  - semi–real time system; with processing occurring after user session is terminated.
  - or real–time system; where all events are processed immediately, while the user is still online.
  - the speed and amount of audit data produced.
  - the efficiency of the filtering or reduction processes.
  - amount of storage required (and what percentage of trail is maintained).
  - ability to re–create events.
  - security of the audit data (examine the possibilities for tampering with audit data).
  - the level and number of audited events (ie. kernel, system, user level auditing).
  - choice and reasoning of auditing selected events.

- *Sophistication of Detection Methods:*
  - whether threshold based detection mechanism.
  - profile based statistical analysis. (anomaly detection)
  - rule–based penetration identification. (misuse detection)
  - the accuracy of detection mechanisms.
  - amount of historical data and training required before attempting detection.

  Whether has the ability to detect

  – failed login attempts.
  – system modifications.
  – data browsing.
  – unusual behavior.
  – user awkwardness.
  – changing user working areas.
  – different types of attacks (single user or collaborating users).
  – gradual deviation in overall user behavior (towards bad behavior).
  – aliasing.
  – sporadic user behavior.

- *Reporting Capabilities:*
  - whether timely.
  - level of detail in warning messages.
  - detail of running status.
  - ability to interact and probe system status.

- *Practicality Issues:*
  - amount of change required to system (kernel hacking, etc).
  - effect of the system on users, ie. additional requirements placed on users.

  Ability to provide effective countermeasures:

  - false positives and false negative rates.
  - capability of system to prevent actions.

  Ability to locally modify the system to own requirements:

  - can it be fine tuned.
  - can new techniques be easily incorporated.
  - portability (can a number of heterogeneous systems be easily monitored).

  System Requirements:

  - whether on host or standalone monitoring host.
  - CPU, Disk and Memory requirements.
  - level of required administrator/user sophistication.

It will also become clear that while there has been considerable research in the field (10–15 years) many of these systems still exhibit some fundamental problems due to the difficult nature of detecting intruders. This is not to say that these systems lack sophistication (actually it is quite the reverse, many of these systems are highly sophisticated), but rather that there is a long way to go before we find systems that are capable of real–time detection and prevention of sophisticated attacks in highly networked, distributed multi–user systems.

The following chapter will look at the basic components of these systems. Providing more detail into the components that go to make up these systems.

2.1 Background: Intrusion Detection Systems

J.P Anderson 1980 [9].
The earliest work on intrusion detection. Anderson proposed the idea of detecting of anomalous behavior by examining audit data. Anderson's early work focused on developing algorithms that could be used for offline security analysis of the system audit trail.
Anderson categorized the threats to a computer system (see Section 1.4.1) and specified how the some of the threats could be detected from audit analysis. He proposed that external penetrators could be detected by analysis of failed login attempts. Internal penetrators could be detected by monitoring failed access attempts to files, programs or other resources. To detect masquerading users he suggested observing users normal activity, reasoning that a masquerading user would have activity patterns significantly different from the real users established normal activity patterns. No suggestion was given how to detect misfeasors or clandestine users.

D. Denning, P. Neumann 1985 IDES model [38].
The Intrusion Detection Expert System model was the first work on providing an intrusion detection framework. The IDES model defined:

- *System Independent Audit Record Format* - allows for IDS to monitor multi–vendor systems. System audit records would be converted to this format before passing to the inference (core decision component).
- *Profiles* - to characterize the observed behavior of subjects of the system. Profiles defined in terms of metric and statistical model. Profiles of Login and Session Activity, Program Execution, File Access Activity. eg. metrics: event counter, interval timer, resource measures.
- *Statistical Inference Engine* - determines whether a new observation in user activity (audit record) is abnormal with respect to the users profile (normal behavior). Mean and Standard deviation model, Multivariate model (correlations between two or more metrics), Markov Process Model (state-transition probabilities), Time Series Model (observations of inter–arrival times of records).
- *Anomaly Records* - records generated when anomalous activity detected (statistical engine, or activity rules). Records specify the event, time–stamp, and the profile field that it recorded as abnormal.
- *Activity Rules* - rules that specify action(s) to be taken when an audit record or anomaly record is generated. Eg. notify SSO of record, update profiles, generate new profiles (from templates).

Sytek 1985-86 [93].
Sytek was also carrying out work in developing a tool to perform automated analysis of system audit data. They were the first to develop a tool that performed simple intrusion detection, additionally they produced data to show that intrusion detection was possible by analysis of the audit trail. Sytek developed a tool that ranked user sessions by their suspiciousness. The tool was based on pattern

recognition theory and algorithms were developed that defined expected behavior as functions of audit record fields. The expected values were determined by training the algorithms on previous audit data sets. The algorithms were tested with normal and abnormal usage to test the effectiveness of the audit fields. For a set of 12 audit features the pattern classifier detected all the simulated intrusions but had a high false–alarm rate (40 - 70%).

A chronology of system developments:

- *Saturne* `LAAS/CNRS - INRIA` 84 [40, 41, 50]

  Keywords: fragmentation–scattering, intrusion tolerance, saturation, fault–tolerance

  Experimental Testbed: Bull SPS7's (MC 68020), Delta-4 experimental atomic multicast network, SPS9/600

- *SRI* 85-86 xref [93]

  IBM MVS/VM

- *NAURS - Network Auditing Usage Reporting System* `SRI International` 85-87 xref [93]

  Terminal Access Controller (TAC) Acess System - SRI-NIC MILNet/Arpanet

- *MIDAS - Multics Intrusion Detection and Alerting System* `NCSC` 86 [138]

  Keywords: anomaly detection, misuse detection.

  Dockmaster Honeywell DPS-8/70 Multics (B2), Midas Symbolics Lisp Machine, Military Gateway & Mailsystem, 1200 users.

- *Discovery* `TRW Information Services` 86 [167]

  Keywords: data driven, customer profiles, pattern recognition, anomaly detection.

  IBM 3090s, MVS, Credit Database, dial-up access, 400,000 inquires/day, 120,000 customer access codes.

- *Clyde Digital Systems Audit* `Digital` 87 (xref) [93]

  DEC VAX/VMS

- *IDES/NIDES - Next–Generation Intrusion Detection Expert System* `SRI International` 87-94

  IDES [53, 54, 98, 93, 94, 96, 95, 97, 101, 102, 103, 104]

  NIDES [75, 76, 8, 73, 72, 7, 99, 100]

- *Wisdom & Sense* `LANL` 87 [173, 90, 63, 62]

  Keywords: probability theory, anomaly detection, misuse detection, decision trees.

  IBM RT 6151-125 + Advanced FPU AIX 2.1, DEC VMS

- *CMW -Compartmentized Mode Workstation* `MITRE` 87 [123]

  Keywords: audit security, data reduction, data compression.

  BSD 4.2

- *NIDX - Network Intrusion Detection Expert* `Bellcore` 88 [21]

  Keywords: anomaly detection, misuse detection, knowledge base, IDM.

  UNIX SVR3.1

- *Haystack* `USAF, Tracor Applied Science` 88 [144]

  Keywords: anomaly detection, threshold detection, tagged events, pattern matching.

  Unisys (Sperry) 1100/60 OS/1100, Intel 286 Z-248PC, Oracle DBMS, CLIPS Expert System Shell

- *ISOA - Information Security Officer Assistant* `Planning Research Corporation (PRC)` 88-89 [178, 105]

  Keywords: network & host monitor, anomaly detection, misuse detection, I&W model.

  Sun 3/260 SunOS 4.0.3, IBM AT AIX

- *NADIR - Network Anomaly Detection and Intrusion Reporter* `LANL` 89 xref [106]

- *Computer Watch* `AT&T Bell Laboratories` Sept 89 [105]

  AT&T System V/MLS (B1)

- *Minos* `CCSR ADFA` Feb 91 [115]

- *NSM - Network Security Monitor* `UC-Davis` 90 [61]

  Keywords: lan security monitor, network traffic analysis, host–host activity.

  Sun 3/50.

- *DIDS - Distributed Intrusion Detection System* `UC-Davis, LLNL, USAF-CSC, Haystack Labs` 91 [145]

  Keywords: network intrusion detection, heterogeneous network, agents.

  SPARCstation, C2 audit.

- *IDA - Intrusion Detection Alert* `Motorola` 91 [118]

  Keywords: anomaly detection, forward chaining, knowledge base, secure audit.

  IBM MVS mainframe, ACF2 audit, ADS knowledge base.

- *USTAT - Unix State–Transition Audit Tool* `UC-Santa Barbara` 92 [124, 70, 125, 69]

  Keywords: penetration identification, state transition analysis, knowledge base.

  SPARCstation 1, SunOS 4.1.1, SunOS C2 BSM.

- *SecureNet, SecureNet II* `RACE SecureNet` 94 [152]

  Keywords: cryptographic, neural networks, intent specification languages, expert systems.

  Integrated Broadband Communications (IBC) network environment.

- *NID - Network Intrusion Detector* `LLNL` 94

  Keywords: ethernet monitoring tool, session isolation, replay capabilites.

  Available DoE and DoD only.

- *ID - Polycenter Security Intrusion Detector* `Digital` 93

  Keywords: case–based, agents, countermeasures, integrated system.

  SunOS, OpenVMS, Ultrix.

- *Stalker* `Haystack Labs` 92-94

  Keywords: proprietary signature database, audit control.

  SunOS 4.1.x, Solaris 2.3, Trusted Solaris 1.1 (CMW).

- *ASAX - Advanced Security audit trail Analysis on uniX* `Faculte Universitaire Notre de la Paix & Institut d́Informatique` 94 [55, 57, 112, 58]

  SunOS 4.1.1, Ultrix 4.3, SINIX-L C2000 5.41.

2.2 IDES - Intrusion Detection Expert System

The Intrusion Detection Expert System (IDES) was developed by SRI International over many years. An initial prototype system was developed for Sun/2 and Sun/3 systems to monitor a DEC 2065 running SRI's modified version of TOPS-20. IDES was based on a culmination of earlier work at SRI and Sytek. The earlier work and the Intrusion Detection Model[38] framework were the basis for the initial IDES prototype system [101]. This early prototype system, has been modified over many years to incorporate new and more sophisticated detection techniques, interfaces, scaleability. Additionally it was later migrated from an Oracle relational database system using Pro*C, C and SQL on IBM/DEC/Sun systems with SunView graphical interface environment to a C based Sun Unix environment using an object–oriented X graphical interface library (libiui).
Some of the specifications for the IDES prototype found in the IDES documentation were:

- to provide a system independent mechanism for real–time detection of security violations (whether outsiders attempting to break into the system or insiders who attempt to misuse their privileges).
- to provide mechanisms for summarizing and reporting security violations as well as detecting intrusions that cannot be detected by the access controls (because they are circumvent the controls or exploit a deficiency in the system or its security mechanism).
- audit format should be designed to be flexible. In which a protocol should ensure timely and secure audit delivery.
- initially designed and developed to monitor single target host, but architecture should easily allow support for any number of heterogeneous target systems.
- based on principal of profiling users behavior (historical data) to describe their “norm” behavior. They reason that detecting calculated deviations from this historical user (and group/system) norm would indicate a masquerader or misfeasor.

The following requirements were specified for IDES:

1. *soft real–time processing requirement* - should be able to detect anomalous activity as, or soon after it occurs.
2. *reliability requirement* - should continue to detect anomalous activity despite partial failures on the prototype, however with a proportional degradation in system performance.
3. *scaleability requirement* - should be able to accommodate increased amounts of activity by proportional hardware addition, without compromise in above requirements.

IDES employs several approaches for detecting suspicious behavior (these are the anomaly detection systems), currently IDES defined two subsystems:

1. *Statistical Anomaly Detection* - detects masqueraders by observing departures from established patterns of use for individuals. Achieved by keeping statistical profiles of users past behavior. Additionally, IDES monitors certain system–wide parameters, such as CPU usage and failed login attempts and compares them with historical established system "norms".

2. *Expert–System Anomaly Detector* - detects known intrusion scenarios that exploit known system and security mechanism deficiencies. By using expert system rules that characterize suspicious activity that are based on knowledge of past intrusions, known system vulnerabilities or installation specific security policy.

   The expert system was a later development, not available in the initial IDES prototype system.

2.2.1 IDES Prototype Overview

The initial IDES prototype[101] (see Figure2.1) was only capable of monitoring a single target system. The prototype featured only the statistical anomaly detection engine. A Oracle relational database was used to manage the system and user profiles, and allowed the system administrator to query the database using the SQL query language. The system administrator could also view the various types of anomalies graphically (or graph various user statistics).

Audit Data

Only a few intrusion detection measures were implemented in the initial prototype system (Discrete measures: time and location of login; Continuous measures: connect time, CPU time, I/O activity and protection violations.). Audit data was collected from the target system converted into IDES audit record format and encrypted to be transmitted to the analysis system. As outlined in the Intrusion Detection Model[38], IDES audit records consisted of:

```
<subject, action, object, error--code, resource--info, time>
```

where,

- *Subject* - uniquely identifies initiators of actions, and location of event.
- *Action* - operation performed by subject on object. (eg. read, write, execute, login, logout).
- *Object* - receptors of actions, resources managed by the system. (eg. files, commands, devices).
- *Error–Code* - indicates any errors that occurred when subject attempted action on object. (eg. write–violation, stack overflow, page error).
- *Resource–Info* - report the resources used. (eg. cumulative connect time, CPU time, i/o activity, protection violations).
- *Time* - date and time which event occurred.

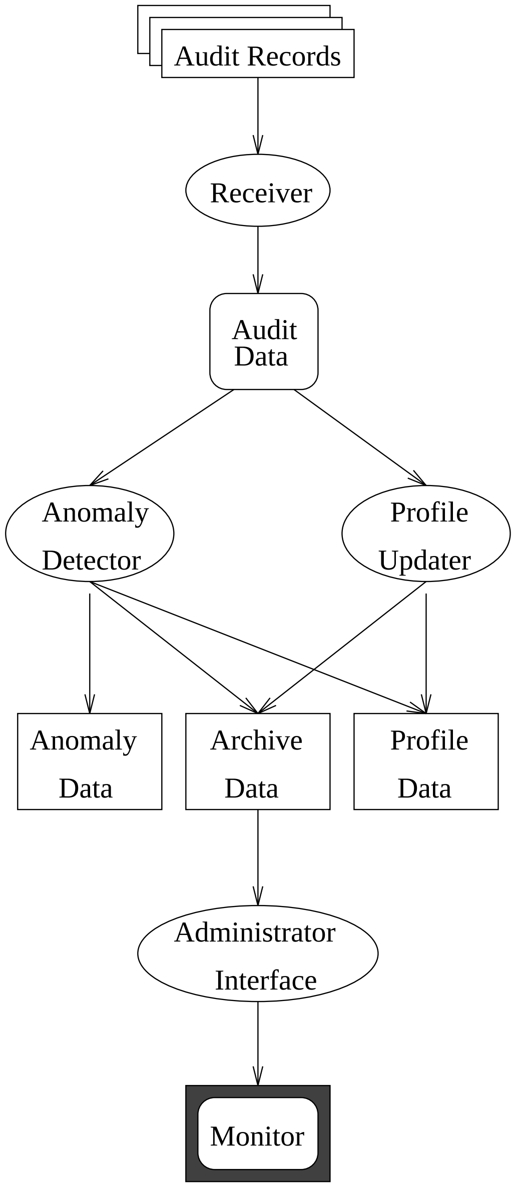

Figure 2.1: Structure of IDES Prototype.

```
ex. copy game.exe to <library> game.exe
<Smith, execute, <Library>copy.exe, 0, CPU=00002, 11058521678>
<Smith, read, <Smith>game.exe, 0, RECORDS = 0, 11058521679>
<Smith, write, <Library>game.exe, write-viol, RECORDS=0, 11058521680>
```

IDES Prototype Database

The IDES prototype relied on an Oracle relational database for management of audit data to be processed by the statistical anomaly detector. The database contains the following data;

- *Audit Data* - valid decrypted unprocessed audit records.
- *Active Data* - the accumulated activity data for each user. Reset at end of each profile update period. Periodically used to update the profiles for various intrusion detection measures.
- *Archive Data* - processed audit records and files. Processed audit records are periodically stored to new files, with files being regularly backed up to tape.
- *Profile Data* - consists of intrusion detection measure (idm) tables. Each table contains (for each user) a value defining normal behavior (based on past behavior) a threshold that defines anomalous behavior for the measure profile update period - at which the statistics defining normal behavior are updated and thresholds recalculated (using recently observed behavior - active data).
- *Schedule Data* - table containing records for each idm; Each record consists of an update period for the associated profile table, a reset period for the associated active data table, the last profile update time and a initial threshold value.
- *Anomaly Data* - set of tables for each type of anomaly; Containing the anomaly records that have occurred for that type. Generated when users current behavior (active data values) deviate abnormally (measure threshold) from the user normally defined behavior (users normal behavior value in profile table for idm).

IDES Prototype Processes

- *Receiver* - receives (IDES transport protocol), decrypts, parses and verifies each audit record from target system, acknowledging valid or invalid audit record. Each valid audit record is then stored to be processed (audit data table).
- *Anomaly Detector* - retrieves audit record (from audit data), updates each record for the user (active data) for the relevant idm. Then the current audit record is compared with the users profile (profile data normal behavior) for the related idm. If anomalous (anomalous threshold), then a anomaly record is generated (anomaly data). Finally removes the audit record (from audit data), storing in archive data.
- *Archiver* - periodically backs up processed audit records (archive data) to external files.

- *Profile Updater* - at the end of each idm's profile period (update period), updates profile table from corresponding active data tables. Then zeros the associated active data. Also recomputes probability distributions (with decay) for each affected profile record.
- *Active Data Resetter* - periodically (reset period) updates the active data. Calculates the total activity recorded during that period, and increments number of periods (which is used by profile updater to recompute new probability distributions) and resets current activity field to zero. (eg. reset period for shift, and location is 24 hours, connect time is time of each session).

IDES Prototype Administrator Interface

The Administrator Interface using SunView window system, allows time–varying graphical displays of target system activity with "zoom" ability using inbuilt database queries. Features:

- *Status Monitor* - displays current IDES status; rate of receiving, rejected and processing audit records.
- *Anomaly Monitor* - displays detected (present or previous) anomalous behavior for particular idm or for all idm's.
- *Query Monitor* - allows IDES administrator to query the IDES database using in–built or ad–hoc queries.

In addition to the prototype system being developed at SRI, a version of IDES was being developed to monitor the FBI's Field Officer Information Management System (FOIMS) running on a IBM mainframe system.

2.2.2 Experimental Results

No data was presented for the preliminary experiments to validate the IDES prototype system. However it was claimed that the system was able to detect anomalous behavior with respect to connection time, shift of login and location of login measures. That is when an account on the target system that had already established a profile with the three measures, was used. An anomalous record was generated when a session that lasted longer than the profile defined normal. The anomaly report was generated usually within a minute from when the user caused the anomaly. This system was tested by analyzing several thousand audit records which were transferred across 2 or more ethernet's and a gateway to the IDES processing system.

2.2.3 Analysis & Conclusion

The IDES incorporates simple statistical measures, they reasoned that they were more effective than the rule–base knowledge based systems (as they didn't require any prior knowledge of actual intrusions). While true, statistical systems require a large amount of historical "normal" audit data to build a useful profiles and thus can exhibit unreliable false negatives and false positives detection rates.

As IDES was implemented on a separate machine, there were obvious security (all audit data delivered over network was encrypted), integration (easier to modify the analysis host without affecting availability of main user system; ie. could modify, train, debug and test, shutdown etc..) and performance benefits (IDES CPU, memory and disk resource requirements wouldn't utilize users resources).

The audit management features allowed secure delivery and archiving over the network. These are important features, as audit trail integrity can be ensured, unlike other systems which audit records remain accessible on the target system for extended periods of time. However no consideration is given towards possible denial of service attacks on the network connection between the target system and the IDES host.

IDES allows implementing monitoring of different host systems (as they only have to translate into IDES format and deliver) due to system independent audit record format and delivery protocol (based on Intrusion Detection Model framework).

IDES was one of a few systems that originally featured a graphical user interface for the reporting and monitoring of the system activity. While analysis was performed within a few minutes, the system lacked any capability to prevent any activity.

While this was only the early prototype system, it did not attempt to provide the 3 system requirements. These came with SRI's later versions of IDES and the Next-Generation IDES (NIDES) which were much more highly sophisticated. Unfortunately could not present the details for these systems due to space limitations (would have to leave out presenting other example systems).

2.3 Haystack

Haystack is an intrusion detection system developed and field tested by Los Alamos National Laboratories (LANL). The initial design and system prototyping[144] was carried out by Tracor Applied Sciences, Inc and Haystack Laboratories, Inc.

2.3.1 Overview

Haystack was developed as a tool for US Air Force computer system security officers (SSO) to reduce voluminous audit data on the Air forces Unisys 1100/60 mainframes which is a standard computer platform in all bases. OS/1100 is a B1 TCSEC rated operating system that runs on the mainframes, however the systems are not running at the B1 rated level. In this environment, external networking is limited to SSO controlled dial–in/dial–out. Therefore Haystack has no requirements for monitoring network activities. Besides any external access can be treated ideally to an insider attack, once the user has access to the system. In [26] it was reported that 80% of computer crime carried out in US Air Force computer systems was by trusted employees.
The goals of Haystack were:

- enable a computer security policy to be enforced by improving the ability to detect and respond to security policy violations.
- develop a software solution that conforms to POSIX and ANSI standards.
- enable SSO to monitor large volumes of raw audit data by summarizing and reporting events deemed suspicious.

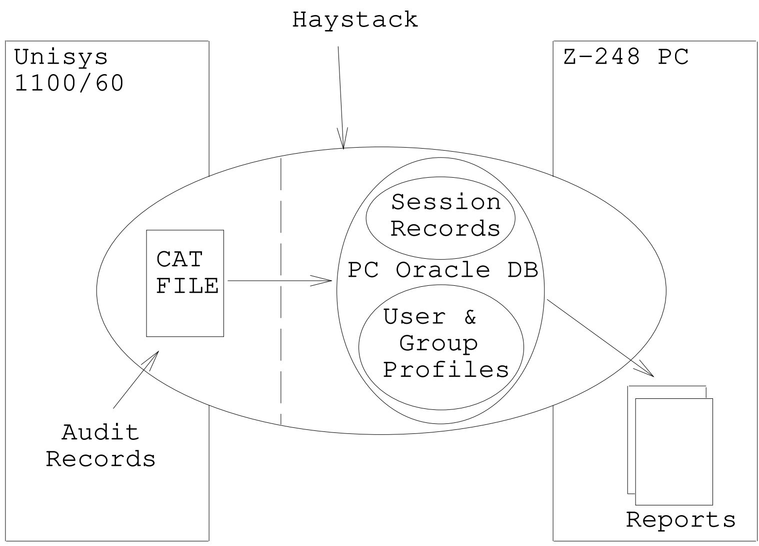

Figure 2.2: Haystack System Diagram

Haystacks approach to intrusion detection was based on the intrusion detection model presented by Denning[38]. The Haystack process shown in figure 2.2 is separated into the following components:

- **Audit Data Collection**
  Haystack audit data is collected on the Unisys (Sperry) 1100/60 and transformed into a Canonical Audit Trail file format (CAT file). The audit material is transferred by tape to a physically secure Z-248PC (Intel based 80x86 machine running MS-DOS) for analysis. Optionally the SSO may select particular users, or particular time intervals for analysis instead of the entire audit trail.

- **CAT File Processing**
  The audit records are read from the CAT tape, any obvious anomalies (see following section) are logged and a new session history record is created for any users appearing in the file. The user session history is cached while processing to improve performance by reducing access to the database on disk. Any security related events that are detected while processing are written to a separate report for the SSO to examine (see Reports section).

- **Session History Analysis**

  Haystack maintains a database of user and user group profiles in an Oracle Database (PC Oracle - a SQL based database management system). The profiles correspond to the security policy which define user and group privileges and capabilities. Haystack looks for misuse in the following ways.

  – *Pattern based analysis* Pattern matching on audit events fields is used to select important events that occur in the users session. The following two mechanisms for selecting audit records are implemented:

    1. Modify Events - successful and unsuccessful events that modify system security are reported. These include modifying access controls, uid and gid.
    2. Tagged Events - any system subjects, objects may be tagged by the SSO for special monitoring. Tagged subjects is essentially marking a particular user to be logged in detail. All user audit events for the session will be reported to the SSO. Similarly when a tagged object is accessed by the user the event is reported to the SSO, optionally when the object event is detected it may turn on detailed auditing of the user (subject).

  – *Statistical based analysis* As in the pattern based analysis, there are two methods of statistical analysis.

    1. Cumulative Weighted Multinomial - by comparing the user's session behavior with the expected ranges of behavior from the user's security group profile. These measures reveal the degree that the user's behavior deviated from expected values, and a "suspicion quotient" is produced. If any user's quotient is outside the expected range, then the entire user session is reported to the SSO. This test is used to identify intruder that isn't behaving similar to the authorized user, or when the authorized user's usage is suspiciously different from the user's normal usage.

2. Wilcoxon–Mann–Whitney Rank Test - by comparing the user's session behavior with previous sessions. This nonparametric test detects differences in the median of the underlying distributions (the user's previous session profiles and the group profiles). This test is used to detect a user that is trying to adapt profiles over time so that normal behavior trends to unauthorized usage patterns.

- **Database Manipulation**
  The SSO has several SQL database applications to assist in controlling the Haystack database. User and group profile editors, session history editor and an editor to maintain the expected behavior and relative weights for each group. Additionally the SSO may develop new applications or use "ad hoc" queries against the database. It is expected that central development of Haystack will continue so that individual SSO will not be involved in extensive system tuning.

- **Reports**
  Processing and analysis of the CAT file results in the creation of two reports for the SSO to monitor. These reports are essentially a summary of the usage of the system. The two reports produced are.

  1. *Summary Report* - lists new users that had profiles automatically created by the Haystack system. Lists users whose " quotients" exceeded the group threshold. It also contains an overview of the system behavior, showing the number of sessions, total number of security events, and number of tagged object accesses.
  2. *Detailed Report* - lists detailed anomalous events detected for each user. Including tagged object accesses, and list of events of tagged subjects.

2.3.2 Experimental Results

No experimental data or results was presented in [144]. However it is noted that Haystack CAT files contain approximately a million audit events per week. And 0.5% of these audit events are finally reported to the SSO.

2.3.3 Analysis & Conclusion

The lack of a detail description the system makes specific comparisons between systems difficult. In addition the design goals and system requirements are quite often very different, making a system comparison futile.
From the report it would appear that some of the goals or capabilities of the system were rather extravagant (ie. would be nice to have features) when compared to the system specifications.
Compared to the IDES prototype, Haystack appeared to concentrate considerable more on implementing decision components. While providing very few features in audit management and no GUI (only producing summary reports). Also the audit records were considered to be independent (Event horizon of 1) which is a more simplistic view than taken in IDES.

Large reduction in the amount of audit data presented to the SSO, however no statistics support how efficient this reduction process was, in terms of rates of failing to report important events.
Limited details about the contents of profiles, or audit records suspected that slightly more measures defined than the IDES prototype system. (from details revealed in figure 1 of paper).
Allowed multiple analysis techniques however lacked detailed explanation of statistical and pattern based analysis. Only that uses Wilcoxon–Mann–Whitney Rank Test and Cumulative Weighted Multinomial for statistical analysis. Also suspect that pattern based matching syntax is similar to the syntax described in Denning IDM, on which the system was also based.
A major limitation (though obviously not critically important to the project requirements) was that it is a batch based system, however was suggested that possible to develop a real–time system.
Haystack system security relatively good due to fact that little networking abilities and analysis machine was physically secure. Also OS guarantees audit trail is generated, although there is no discussion whether it is possible for auditing to be disabled or possibly deleted by users of the system.
Untimely analysis of the audit data due to the manual tape transfer process, which also means audit could possibly be tampered with. System lacks any ability to abort abusive user activity.

2.4 STAT/USTAT - Unix State Transition Analysis Tool

The Unix State Transition Analysis Tool is a Unix[1] specific prototype Intrusion Detection Tool [70, 124]. USTAT is based on the unique "State Transition" model presented in [124] as STAT, State Transition Analysis Tool.

2.4.1 Overview

USTAT monitors critical actions in the users audit trail (in real-time) and determines whether these actions lead to the successful completion of a penetration rule. The penetration rules are defined in terms of states and state transitions. A state is defined as user and/or environment data stores at a particular time in the system. A state transition is an action that causes change from one "state" to another. For a successful penetration identification, USTAT must match the change from an initial state to the final state by a sequence of actions.

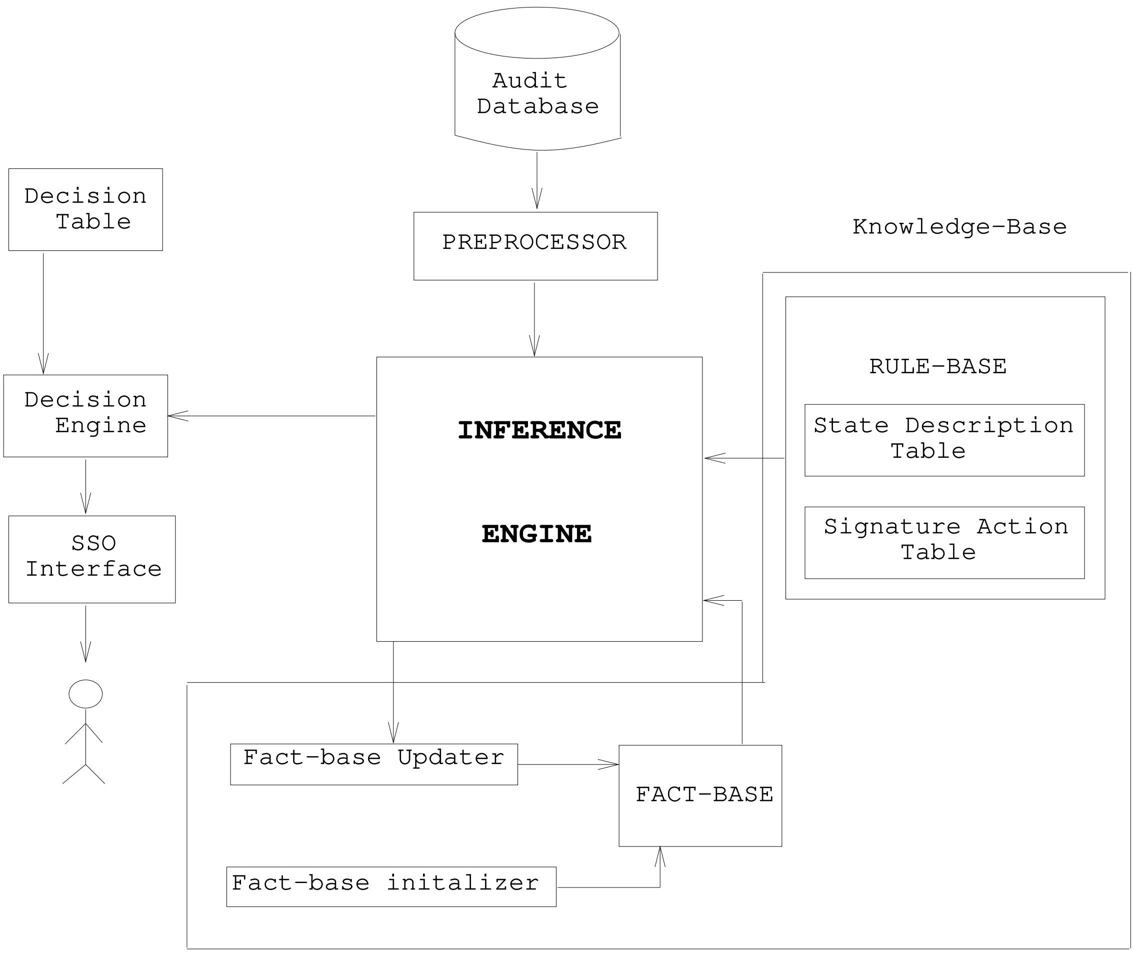

Figure 2.3: USTAT System Diagram

[1]developed for SunOS 4.1.1

FILESET	*CHARACTERISTICS*
Fileset #1	Restricted read files
Fileset #2	Restricted write files
Fileset #3	Files authorized to read Fileset #1
Fileset #4	Files authorized to write Fileset #2
Fileset #5	Non-writable system executables
NWSD	Non-writable system directories
Hardlink	System hardlink information

Table 2.1: USTAT Filesets

The USTAT system2.3 is comprised of:

- **Preprocessor** for reading, filtering and reformatting the C2 audit records before passing them to the inference engine for processing. The preprocessor allows portability of the system by processing audit records into generic USTAT specific Audit Format.

  Audit Format: <Subject, Action, Object>

  Subject:- <User ID, Effective User ID, Group ID>
  Action:- <Action, Time, Process ID>
  Object:- <Object Name, Permissions, Owner, Group Owner, Inode #, Device #, File System ID, Target>

  Out of 239 possible auditable events only 28 are used by the preprocessor, being classified into 1 of the 10 actions groups.

  (Read, Write, Create, Delete, Execute, Exit, Modify_Owner, Modify_Perm, Rename, Hardlink)

- **Knowledge Base** comprised of:

  – **Fact–Base** - groups files and directories into 7 filesets (Table 2.1) based on characteristics of the file/directory.

    A fact base *initializer* is used to initialize the fact base. The fact–base *updater* maintains the fact base, and keeps it upto date with the inference engine.

  – **Rule–Base** - State transition information is stored in 2 files. These files are used by the inference engine to match audit records to penetration scenarios defined in terms of state diagrams. Again, the state diagrams consist of “states” or state assertions, and “actions” the necessary operation to move from one state to another. See figure 2.4.

    1. State Description Table - stores the state assertions. The state assertions are defined using the following functions

       (a) **name**(*file_var*)=*file_name*
       evaluates TRUE if the file_var matches the file_name.

       (b) **fullname**(*file_var*)=*full_path*
       evaluates TRUE if the file_var matches the pathname.

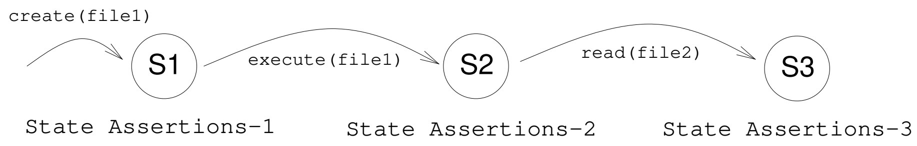

Figure 2.4: Hypothetical State Transition Diagram

(c) **owner**(*file_var*)=*user_id*
evaluates TRUE if the owner of *file_var* is the *user_id*.

(d) **member**(*file_set, file_var*)
evaluates TRUE if *file_var*is a member of the *file_set*.

(e) **euid**=*user_id*
evaluates TRUE if the effective user id of the subject is *user_id*.

(f) **gid**=*group_id*
evaluates TRUE if the group id of the subject is *group_id*.

(g) **permitted**(*perm, file_var*)
evaluates TRUE if *file_var* permission bit is *perm*.

(h) **located**(NWSD,*file_var*)
evaluates TRUE if *file_var* is in any of the directories from the NWSD fileset.

(i) **same_user**
evaluates TRUE if the subjects of the last two actions are the same.

(j) **same_pid**
evaluates TRUE if the process id of the last two actions are the same.

(k) **shell_script**(*file_var*)
evaluates TRUE if *file_var* is a shell script (ie. #!/bin/sh).

2. Signature Actions Table - stores the state actions. The actions are defined using the following functions

(a) **read**(*file_var*)
(b) **write**(*file_var*)
(c) **create**(*file_var*)
(d) **execute**(*file_var*)
(e) **exit**(*file_var*)
(f) **delete**(*file_var*)
(g) **modify_owner**(*file_var*)
(h) **modify_perm**(*file_var*)
(i) **hardlink**(*file_var, file_var*)
(j) **rename**(*file_var, file_var*)

- **Inference Engine**

The inference engine uses the currently defined facts and rules at the particular time (ie. defined in the rule–base; State Description and Signature Action Tables) then tries to match a series of audit records to state transition diagrams.

An inference engine table is used so that multiple scenarios can be followed simultaneously[2] for every user on the system.

The table therefore contains all the possible penetration instantiations that are under construction. To follow multiple scenarios simultaneously, when the next state is achieved (for a particular penetration state diagram) the row in the inference table is duplicated and the next state cell is marked in the duplicated entry. State hypothesis are only removed from the inference table when an assertion can no longer hold (ie. one of the objects has been removed) or when the final state is matched. If the final state is matched a message is sent to the decision engine.

- **Decision Engine**

  The decision engine is responsible for informing the SSO that a compromise has been achieved. When reporting a compromise the decision engine consults an internal table for the corresponding message. An identifier that indicates the state transition diagram that was compromised. The filenames of any files involved in the instance, and the users id and effective user id for the user who performed the last signature action are displayed so that the SSO can make the necessary decisions to prevent further attacks.

2.4.2 Experimental Results

Some of the experimental results are produced in [69].
3 different audit intensive processes (CPU intensive produce less audit) were performed.

1. Starting X–Windows produced 161 Kb of audit data, taking 03:38 minutes with USTAT running, a difference of 02:51 compared to when auditing is disabled. (ie. 00:47).

2. Compiling USTAT 143 Kb, taking 03:58 with a difference of 2:58 (1:00).

3. Running COPS 4,570 Kb, taking 01:20:45, difference of 01:01:55 (18:50).

Some of benchmarking results of processing various runs of audit records with 12 rules in the rule–base (see below), with no major CPU load on the machine running USTAT is in Table 2.2.
Where
Data - size of raw audit in kilobytes.
Audit - the actual number of audit records.
USTAT - the number of records that are used by USTAT.
A/U - ratio of Audit/USTAT.
Time - time elapsed to process these records.
K/s, A/s, U/s - number of kilobytes, audit records, and USTAT records processed per second.

[2] This mechanism also allows for the detection of multiple users that are working together to penetrate the system.

Max - maximum number of rows reached in the inference table during processing.
Cur - number of rows in the inference engine table at the end of processing.

Data (Kb)	Audit (#Recs)	USTAT (#Recs)	A/U Ratio	Time (mm:ss)	Kb/s	A/s	U/s	Max	Cur
667	5,584	2,288	0.409	0:59	11.5	95	39	62	56
1,526	13,916	3,245	0.233	1:26	17.8	162	38	–	–
5,127	47,669	13,865	0.291	6:27	13.3	123	36	76	39
10,525	94,033	28,576	0.304	14:20	12.5	231	34	77	39
51,259	480,643	213,965	0.445	1:03:41	13.4	126	56	150	140

Table 2.2: Benchmark results for various audit collection runs.

The 12 state transition diagrams encoded in the rule–base test the following scenarios;

1. creation of a symbolic link to suid shell script, in which link name is -char (ie. -i, option that specifies interactive shell be created).
2. creating world readable mail spool files.
3. suid shell script that invokes executable binary, where the binary selected to run is substituted (by modifying $PATH) to one that exec's an interactive shell.
4. lpr -r removing of files not owned by the attacker.
5. execution of other users suid shell scripts.
6. reading of restricted files (by programs not specified to read restricted file set).
7. as in 6 except writing.
8. overwriting other users files.
9. creating suid programs.
10. overwriting executables in directory set (non system writable directories).
11. as in 10 except creating.
12. deleting other users files (even though file permissions allowed it).

2.4.3 Conclusion & Analysis

A unique approach, where computer penetrations can be represented in a clear and concise way, as state transition diagrams (unlike other ad hoc methods). Allows for simpler, more intuitive rule generation, which are easier to analyze for rule reductions or simplifications. Also allows a simple GUI editor to developed that would allow rules to be constructed graphically.
First system to have the ability to detect semi-sophisticated attacks (co–operative attacks), however only a single co–operative test has been developed so far.

Whereas current systems pattern match audit data, where audit fields are tested, expecting to match to some facts and rules. STAT focuses on the effects of individual actions (of penetration) have on the system. That is STAT maintains a current state, so essentially audit records are considered dependent and independently. There also exists problems when intrusion steps are executed out of order (as the rulebase only considers scenarios from the current states). So the second last state might be activated first, and the initial conditions and last condition then performed. In effect the intrusion would be missed by the STAT unless each of the transition rules are applied for every audit event. This isn't difficult if the states are considered as mask registers in which particular states can be marked dependent on a single transition (same as reducing long rules so that essentially handles out of order events).

Also this system only detects known intrusion techniques, suspicious activity or local policy restrictions in which rules have been developed. Does not detect masquerading users (unless the masquerading user performs known misusive activity).

Chapter 3

Intrusion Detection System Components

From the systems presented in the previous chapter, we see that Intrusion Detection Systems essentially consist of three components; The collective or audit component, which is responsible for the collection of information about the system usage. This information may require some form of filtering, compression or archiving. The decision or detection component, which deals with the reasoning of the recorded events utilizing some form of stored information (knowledge base) to identify behavior that is anomalous or abusive. Lastly, the reporting or alert component. This is responsible for notifying the administrator of any significant events detected and/or carrying out any necessary countermeasures. This chapter investigates the various components of Intrusion Detection Systems, presenting some of the methods used in the various systems listed earlier.

3.1 Auditing System

The audit system is responsible for collection of records of user/subject activity. Most, if not all current IDS collect additional activity records to that of the standard system generated audit trail. The general audit process consists of generating audit records for necessary events, audit reduction, formatting of audit records (into IDS Record Format), and finally delivery of audit records to IDS decision component.

3.1.1 Audit Record Creation

As most systems produce some form of audit data, it is useful to use this source of information. However most of the audit material generated by the system does not record all the necessary information about the event that occurred, this is due to the fact that most audit systems were developed for accounting purposes only. So modification to the audit facilities is often necessary to select more useful data, so that a better decision can be made whether the event is anomalous or not.

For example, audit records are typically created after some action is completed, ideally it is useful to also audit the commencement of the action otherwise incomplete actions often are never audited. Audit can be produced at all layers of the system:

- Kernel System Calls - Auditing at the kernel level is the lowest level of auditing. Auditing the kernel level can generate a lot of audit data, so usually auditing isn't performed on all system calls but only a selected few. Additionally, a single system call usually isn't detectable as "anomalous", usually a collection of calls will be necessary to describe an anomalous event.
- System binaries & Daemons - Auditing the system binaries is essential for detecting users trying to exploit "holes" in the system security. It is also necessary for creating profiles of user's system activities. Auditing of system daemons also reveals usage patterns of system services and resources.
- Shell - The shell allows a complete trail of the users session command activity.

3.1.2 Audit Reduction/Compression

Due to the large amount of audit material that is generated. It is common for IDS to perform some form of audit reduction. This usually either via data compression of the audit records or filtering only the specific events that are considered necessary for the determination of anomalous activity.

- Audit certain events (defined by event classes). USTAT maps BSD system calls into event classes.

  USTAT: 28 events out of 239 audit types mapped into 1 of 10 action groups < Read, Write, Create, Delete, Execute, Exit, Modify_Owner, Modify_Perm, Rename, Hardlink >.
- Compressing audit data. CMW used Huffman compression routines (`compress`) to compress the audit data. The compression of the audit trail is controlled by a daemon that controls the compression of incoming audit data as well as file switching of the output, compressed audit trails.

3.1.3 Audit Record Format

Commonly the IDS defines a system independent audit record format, this allows the IDS audit collection process to be portable. This allows monitoring of heterogeneous networks by transforming the different system audit formats into the IDS audit format. Monitoring additional environments only requires an additional preprocessor.

- IDES Audit Records:
  < Subject, Action, Object, Exception–Condition, Resource–Usage, Time–stamp >

  Action - operation performed by subject on (or with) object.

  Exception-Condition - denotes any exception conditions raised by the system.

Resource-Usage - list of quantitative elements, gives the amount of some resource (eg. number pages printed, CPU time, I/O units used, elapsed time).

Time-Stamp - unique time/data stamp, identifying the time of event.

3.1.4 Audit Delivery

Ensuring secure delivery of audit material to the decision component of the IDS is crucial to positive detection. If the audit material can be intercepted and modified, or disabled then the user is effectively able to avoid detection (clandestine).

- Haystack's audit trail is collected on Unisys 1100/2200 running OS/1100. A preprocessor converts the target systems audit trail into Canonical Audit Trail (CAT format) and transferred through a serial link or via nine–track tape to a physically separate and protected DOS based machine.

- ISOA receives audit records from a network of Unix systems. The audit data is sent via a UNIX socket in encrypted packets. ISOA receives the audit records, decrypts and verifies them before processing them.

3.2 Decision System (Inference Engines)

The decision component of IDS is concerned with taking audit records (or events) and determining whether the events correspond to anomalous or misusive behavior. Where anomalous behavior is deviation from the users normal usage patterns (ie. masquerader). Or the event corresponds to "known" intrusion patterns, or conflicts with site computer system policies. There are many different approaches to performing the decision analysis. The following represent the major decision component developments in IDS. Some systems incorporate multiple decision components for more effective detection capabilities.

3.2.1 Statistical–Based

Statistical–based systems use statistical analysis of a user's activity (user profiles) which is based on past system usage patterns. Detection of anomalous behavior is achieved by monitoring for behavior patterns that deviate from those defined in the user profile.

- The IDES statistical component was based primarily on mean and standard deviation, and was later expanded (NIDES) into a complex structure that incorporates correlation between measures and time–based decay factors to allow older profile information to be considered with less statistical weight than recent behavior data. This allows more consistent user profiles to be constructed for users who's system usage patterns are continually changing.

3.2.2 Rule-Based

Rule–based systems incorporate known misusive events or intrusion scenarios into a knowledge base. Detection of misusive behavior is achieved by comparing the users audit records against the rule base. If a rule is matched then an alert is generated and sent the SSO.

- Wisdom and Sense analyzes audit records against one or more sets of rules. The rules are organized in a tree forest structure, which reduces storage requirements and enables for rapid searching. The size of the rule forest in Wisdom and Sense is very large (10^4 to 10^6 rule instantiations). However the rule base can usually be searched in 0.05 seconds. The rules are expressed in terms of specific values rather than variables. These rules are automatically generated from historical data, and a rule grade (weight) is calculated/updated and stored with the rule conditions (LHS), for implied conclusion's (RHS). LHS is a series of values, value ranges or computed values based upon series of related records (ie. mean time between some event type) or subroutines returning a boolean. Audit records are compared with the rule forest, a record that satisfies the LHS (ie. value matches or in range) and fires. The RHS determines the expected conclusion for such rules (ie. the RHS is the normal values). An audit record field that violates many conclusions (RHS ie. expected values) is considered anomalous.

3.2.3 State-Transition

State-Transition based systems incorporate known abusive behavior as a sequence of states and state–transitions. Essentially the system monitors the state of the user session, looking for state conditions that specify an abusive state (multiple states are stored for each user session in state description tables). The state–transitions are coded into a rule base. By examining incoming audit events to the state–transition rules it is possible to determine if a change in the session state should be affected. Once a goal state is achieved an alert is produced to notify the SSO of misuse as in STAT[124] and USTAT [70].

3.2.4 Neural-Nets

Learning capability is a characteristic of neural networks. A neural network component allows the IDS to learn the law of user behavior, through the audit trail without explicitly expressing it. The learning algorithm allows the network to follow the user behavior patterns closely and adapt itself to the constantly occurring changes. The network tunes the threshold weights to realize a function. If there are correlations in the input the network will converge to a more stable state. A feature of neural networks is they can cope with fuzzy or noisy data sets.
Neural Network model is based on three hypothesis:

1. User's are creatures of habit. A user will develop a set of preferred commands for working, instead of frequently changing their working patterns.

2. The behavior of the user, through habit follows a stochastic law. The gathering of command names (events), CPU and I/O usage variables creates a multivariate time series.

3. Correlations exist between various measures contained in the audit records.

- The development of the complex statistical engine in IDES/NIDES lead to the SRI developers examining the possible use of neural nets and fuzzy logic due to the slow development times, manageability and complexity of the statistical algorithms.

- In [36, 37] a recurrent neural network component for intrusion detection is examined. In the prototype IDS they combine a neural network coupled with an expert system. The users session audit trail is viewed as a multivariate time series, forming the basis for the neural network component.

3.2.5 Fuzzy-Logic

Some problems identified in other decision systems have suggested that "fuzzy" reasoning should be incorporated into IDS. Theoretically fuzzy logic allows encoding anomalous activity rules in a fuzzy fashion, that is based on rules that have been satisfied so far and knowledge of the user we should be able to weight events to probability of intrusion.

- Retiss [31] is a real–time IDS based on the hypothesis that there exists a correlation between anomalous user behavior and threats. In Retiss the security policy is expressed in a rule–base, additionally behavior patterns that correspond to possible threats are represented. Weight tables are used to define the levels of danger that correspond to each rule. The levels of danger of anomalies are fuzzy combined to express probability of threats.

Two important characteristics of the decision components of IDS are the abilities to:

3.2.6 Pre–empt or Abort

The ability of the decision components to successfully identify anomalous and misusive behavior is a complex problem. Once the decision engine has concluded that an event (or sequence of events) is abusive it would be useful if the system could automatically pre–empt the intrusive event(s) (ideally in the progress of an intrusion pattern) or abort the user session. Most systems are currently not sophisticated enough to perform this feature. It complex issue due to the fact that unless the false–rejection rate is relatively low (approximately 1%) the users are likely to become intolerant of a system that keeps rejecting their processes or session.

3.2.7 Updating

Users behavior patterns change over time, the commands used by a user vary as the user becomes more familiar with the system. A comprehensive system should include the ability to automatically update the user profiles. That is the system should learn the users new behavior patterns as time progresses. This introduces problems in a system that has a high false–acceptance rate. Such that it accepts unauthorized users and learns their behavior patterns for the authorized user. An additional concern with this mechanism is that it is susceptible to users that slowly deviate their behavior towards unauthorized activity. A system following this behavior would accept unauthorized activities as normal user behavior.

3.3 Reporting System

Once an intrusive event is detected most systems opt to inform the SSO of the event. Letting the SSO decide on the desired course of action. In general the following two mechanisms are used to report intrusive events:

3.3.1 Audit Summary Report

A summary report is a list of the critical events that were identified in the previous run, that should be further examined by the SSO. This is common to the simpler systems where the main goal of the IDS is to perform audit reduction with some simple intrusion detection capabilities for selecting events deemed critical.

3.3.2 GUI

A GUI front–end is more common on the sophisticated IDS. GUI's give access to the audit trail records, lists of events that are suspicious and brought to the SSO attention directly (when deemed critical). The GUI front–end allows the SSO to quickly visualize the state of the system and perform ad-hoc queries on the activities of users on the system.

3.4 Distributed IDS

The more complex IDS, especially the systems that attempt to operate in real–time are very CPU intensive (also with large amount of audit data being processed can be very disk intensive). Most systems recommend a separate host for analysis of the host audit trail, some even require the reporting system running on a separate host. This is a simple form of distributing the IDS components, a few systems incorporate more complex distribution of the IDS components over the domain of systems that are being monitored.

- NIDES Granular Lucid (GLU) [103].
- Distributed Intrusion Detection System (DIDS) [145].

Chapter 4

Intrusion Detection System Extensions

This chapter is a discussion of some possible system extensions and approaches that I have developed during the course of investigating other intrusion detection systems. The first section deals with agent based intrusion detection, this idea was based on wanting to be able to provide preventive capabilities, in addition to allowing for a highly configurable system with a sophisticated end user interface. The initial idea of using an agent based paradigm came from articleson work in developing personal digital assistants.

The idea of a tool for automated policy enforcement is due to the experiences of problems in our local environment. The lack of tools to control system security mechanisms and limitations in the standard access controls. Additionally many problems with the current audit systems and the practical limitations of enabling audit. An outline for a concept tool used to automate the process of setting and managing security controls is introduced. By enabling particular features to be detected, we can provide greater control ensuring only authorized access to system services.

Lastly, the idea of universal intrusion detection systems comes from investigating the various intrusion detection systems. These systems were usually designed for very specific purposes (though many systems tried to implement some level of system and task independence). Yet the audit data available to intrusion detection systems has not improved. I reason that for any wide spread acceptance of these systems there needs to be some form of standardization among the IDS community. Even though most intrusion detection systems comprise of the same audit, decision and reporting component structure there has been no effort to standardize these mechanisms (while many systems used the Intrusion Detection Model[38] as a foundation there remains little inter-operability amongst systems/components).

While none of the ideas here have been developed in any way, they have received considerable discussion; being discussed on the intrusion detection systems mailing list and in my workshop presentation[91]. It is my intention to develop prototypes based on these ideas in my further work in intrusion detection.

4.1 Agents

The concept of agents was first purposed by developers working at Apple, and later General Magic. The idea was developed in researching personal digital assistants (PDA's). In these systems, agents were being developed to be responsible for automating handling of tedious computer tasks. The user configures their agent programs to perform repetitive tasks such as filtering mail, organizing meetings, obtaining information or information references by scanning network services and archives. For example, when agents organize meetings. The user specifies the meeting topic, participant list and suitable time period. The agent is then responsible for sending requests to all participants agents, automatically handling responses and arranging for the most suitable time, and finally notifying the user of the scheduled meeting time by updating users computerized daily planner. Agents communicate with various software programs (through special application specific agent commands) to obtain data relevant to the task. Communicate with other agents using user supplied parameters to make decisions (requesting further details if decision is not possible). Essentially agents ensure minimal human intervention after the task is specified. So the user specifies what is necessary and the agent is responsible for carrying out the tedious details of the task.

Agents also seem well suited to intrusion detection. To remove the tedious task of collecting, filtering and human analysis of audit data. They can be programmed to allow different levels of monitoring to be applied based on input data and decisions. Only informing the administrator of important events that were requested.

Agents require a small repertoire of commands with which they can communicate information amongst themselves, allowing them to distribute themselves across the domain to provide both fault tolerance and increased performance by sharing load on the system resources. However in such a scheme it is crucial that the integrity of both agents and data being passed around is ensured.

While no prototype system has been developed so far (nor any proof of concept), some thought has been given to the integrity problem and some of the possible features and potential gains from such a system.

Some of the potential gains of such a system:

- More efficient utilization of system resources, and significantly less intervention required from the system administrator.
- The methodology allows simple components (ie. smaller/task specific) to be designed resulting in simpler reasoning techniques to be integrated. Which allows for direct preventive capabilities, or at least some form of effective countermeasure capability to be included.
- Simplicity of integrating of new agent abilities. So when a new service, program or resource is incorporated in the system. New tasks can be embedded into a new agent to achieve service specific tasks.
- Allows high level primitives to be developed from low level primitives. So that low detail can abstracted from the system users. Therefore allowing less experienced administrators to use the

intrusion detection system. Unlike the many other systems that required vast technical knowledge of the low level processes being performed.

- Modularization and Compartmentalization - components can be designed specifically for system, service, resource or program monitoring tasks. By maintaining task specific profile data allows separate audit trails (or status reports) to be generated in which all audit records consists of similarly related events (task specific audit data).

- Configureability - agents can incorporate different components depending on the requirements of the agent. These components can then be dynamically modified allowing agents to modify there roles when necessary (ie. after (de)-activation of processes). An additional benefit is that agents can be easily implemented for heterogeneous systems, where system independent components such as data structures for profiles and analysis can be incorporated with components with system dependent requirements.

- Prioritization - agent tasks can be assigned with priority weight therefore maximizing efficiency. Such that an agent would contain job list and execute tasks according to the priorities.

- Control - the system administrator using an agent communication protocol could submit status queries or new job specifications to agents. Also inter–agent communication allows similar agents to share information or direct tasks of other agents more specific to task.

- Such a scheme does not have to be a total replacement for the already developed systems. Information from the agents could be passed back to more sophisticated (secondary) statistical or rule–based expert system analytic engines for additional processing and audit storage.

Essentially agents would allow for dynamic audit management, profile management and reasoning, system status queries, basic system configuration and control.
The integrity of the agents depends directly on the system users ability to adversely affect agent operations (CIA):

- Attempting to overloaded agents (denial of service). By creating large numbers of system events. Agents should allow denial of service attempts to be detected earlier (eg. if number records generated in certain period is abnormal then the user session or process can be suspended or terminated).

- Modifying the agents tasks. Attempting to spoof agent jobs as the administrator or another authorized agent (authentication). All jobs can be authenticated using challenge response systems. Or a simpler technique would only allowing registered agents to communicate. Different levels of authentication could be used according to the agents role. Having agents implement self monitoring (or monitor each other) such that an unauthorized user who attempts to communicate with an agent (spoof) would be detected and the request disregarded.

- User can not obtain agent data (confidentiality). Again self monitoring techniques and cryptographic techniques are useful. Also standard operating system access control mechanisms would be used. Agents running at a different level than user process (so a user would have to obtain system privileges without being detected). Additionally agents could be configured to run only on systems within the network domain on which the user has no activity (on detecting any user activity on the agents system, the agent would move to another host).

Thought has been given towards the development of a prototype system. Obviously an object oriented paradigm seems well suited to such a scheme. Using interpreter languages also allows for quick prototyping and are also suitable to dynamic nature system.

1. *Audit Data* - data collected either directly by standard system audit (Though my intentions are to redesign an audit subsystem specifically towards intrusion detection requirements; see 4.3.1). Or by means of call backs incorporated into both system binaries and available in dynamic libraries. Finally components can collect there own data. Such methods include using /proc filesystem, performing network service queries or standard status commands. In addition archive management agents would ensure anomalous (or general) audit management.

2. *Agents* - implemented using a mix of interpreter languages (ie. Perl or TCL), and compiled languages (C,C++). Using an object oriented (OO) style design. TCL and Perl are the likely choice as they feature OO and networking extensions. Critical run–time components can easily be implemented in C for efficiency, if required.

3. *Inference Engines* - in addition to the simple agent rule reasoning (based on specific minimal knowledge that prove to provide efficient detection), event matching and threshold detection techniques. Sophisticated secondary statistical, neural or rule–base expert systems (LISP) could be developed to perform both anomaly and misuse analysis on the specific data forward by the agents (for soft real–time processing).

4. *User Interface* - TCL/TK or Perl/TK well suited. TK component allows for quick user interface prototyping. Later user interface components could be developed in C using Motif.

The specific agents recognized so far are:

4.1.1 Guardian

Guardian agents are assigned to monitor or analyze user activity, such that when a new user logs into the system a new agent is created to monitor all activity until user session (or activity) ends.

4.1.2 Sentry

Sentry agents are assigned to monitor service or resource requests. An example situation would be an NFS server (or client) agent that monitors all NFS requests and responses. The agent would manage data on the NFS server status, types of requests, which requests particular users have performed, the number of requests and privilege violations, any administration tasks. In addition to allowing the agent to prevent requests (either by informing NFS server to disregard request or filtering requests so request never appears) or invoke additional monitoring mechanisms within or in other agents (such as the users guardian).

4.1.3 Hunter

Hunter agents are used collect various information from other agents. An example scenario is that the system administrator wants to query a particular users network activity. So the hunter agent can collect relevant data from other agents or initiate a new monitoring job tasks amongst other agents to monitor the user network logins, effectively tracing all logins over local domain.
The above three differentiate in that guardian collects and monitors user data, sentry for services and resources and the hunter agent performs administrative queries and control. Each agent would incorporate task specific data, monitoring, decision and communication components.

4.2 Enforcing Computer Policy

One of the major problems a system administrator faces is how to effectively control access of the system services and resources. Most often the administrator must rely on access control features of the operating system or individual configuration files to permit or deny access to system resources. However due to the size of some systems currently being administered, in some cases 10,000+ users, it becomes a difficult task to ensure correct access controls for every users. The latest release of Solaris, Solaris2.3 (SunOS 5.3) comes with a new set of system administration features incorporated into a program called **admintool**. The Administration tool is an graphical user interface front end to the Network Information Service (NIS+) tables. This integrated tool allows for the management of several system administration tasks[1] that makes the task of administrating large a network easier. By allowing a single tool to control service configurations for the entire domain.

While this tool, eases the task of maintaining or setting up services on the system, it does not incorporate actual tools to assist in the management of security of the system per se. This is not to say that the increased authentication due to Secure RPC and increased access controls over NIS+ tables isn't an increased security benefit, but rather that there is still a lack for administrating control of the system security.

There is still little direct control over services and general monitoring of system resource usage. Most of these are still text based configurations with limited access control features. The auditing facilities of the Basic Security Module (BSM) allow for very extensive audit capabilities, yet lack any overall interface. Instead requiring the use of multitudes of commands and text based configuration files.

Typically in the university environment (and probably applies to many others) users are required to sign a computer usage policy agreement. Essentially the department and administrators decide on what services particular users are allowed to use (ie. under–graduates have different services, quota's, ... than the graduates). The policy is therefore a means of dictating the correct procedures for using the computer systems, in addition to outlining any penalties for violation of the restrictions.

Usually most services have been provided, but due to general abuse of some services such as ftp and irc, these services are often restricted. Now the system administrator is required to maintain that the restricted services are only available to those who have legitimate reasons for use. Here is my general point; *While it is generally easy to specify a computer policy on paper, it is often difficult (if not impossible) to enforce*. There are a mix of tools available to monitor general services, but generally each administrator tends to re-implement their own solutions, kludges, hacks of these tools. Even when on paper it is quite possible that users will argue that there actions were not in violation of the policy, that the policy was not specified clearly or that they had not realized that it was a policy restriction. And eventually users will find ways around (or to avoid) the restrictions.

So what would be the steps for incorporating a paper based computer policy into a system enforcing tool ?

First will investigate some of the issues involved with implementing such as system:

[1] The Administration Tool consists of a Database Manager, Printer Manager, Host Manager and a User Account Manager.

4.2.1 What are the problems ?

- Many administrators have in excess of 1000+ users, thus there are limitations on their ability to ensure and maintain correct access controls due the magnitude of files, difficulties in specifying access controls due to granularity restrictions, and problems with default settings.
- Difficult to control and monitor system services. For example how do we restrict ftp or irc access.
- Enforcing restrictions is difficult. Users tend to find ways to circumvent the restrictions.
- Every system administrator tends to re-develop their own methods: usually some kind of service/program wrapper.
- Often there is inadequate support staff for the system and maintaining general working order is a higher priority than the security concerns.

Now many users have well defined domain in which they are expected to work, so it should be possible to specify a policy for users, groups, services and resource usage.

4.2.2 What is a policy ?

Policy is usually in the form of a paper document that specifies in some detail what is acceptable usage of the computer facilities. Often it is difficult to specify detailed requirements for users. In turn, a user can't be expected to memorize every possible restriction (and may be difficult to specify every individual restriction). It would be useful if users could rely on the system to indicate when they are attempting actions that are in violation to policy and warn them (in addition to stopping there actions).

4.2.3 How would we formalize a policy ?

How do the computer managers, department staff and system administrators specify what is allowed and what isn't when they maybe unsure of exactly what services, applications and resources are available on the system. Therefore some automated process for identifying these services on the system would also be beneficial. Once the possible services and resources are known, then a policy has to be resolved for the individual users and groups.

NOTE: It may be useful to specify restrictions on a service/resource basis rather than establishing restrictions for each individual user. As it easier to configure a file transfer service to restrict all and allow certain users/groups than to specify access controls for each user/group.

This may appear as it is just setting access controls, but the idea here is to maintain complete controls over all resources associated with a service. In many cases maintaining individual uid's and gid's is to cumbersome.

Now there only remains some implementation issues (which I will not go into here) but they are: How to represent the computer policy in the computer system ? How to detect policy violations ?

4.2.4 Policy Formalization and Enforcement Tool

The initial design for a tool is called "P-FORMENT", as shown in figure 4.1.

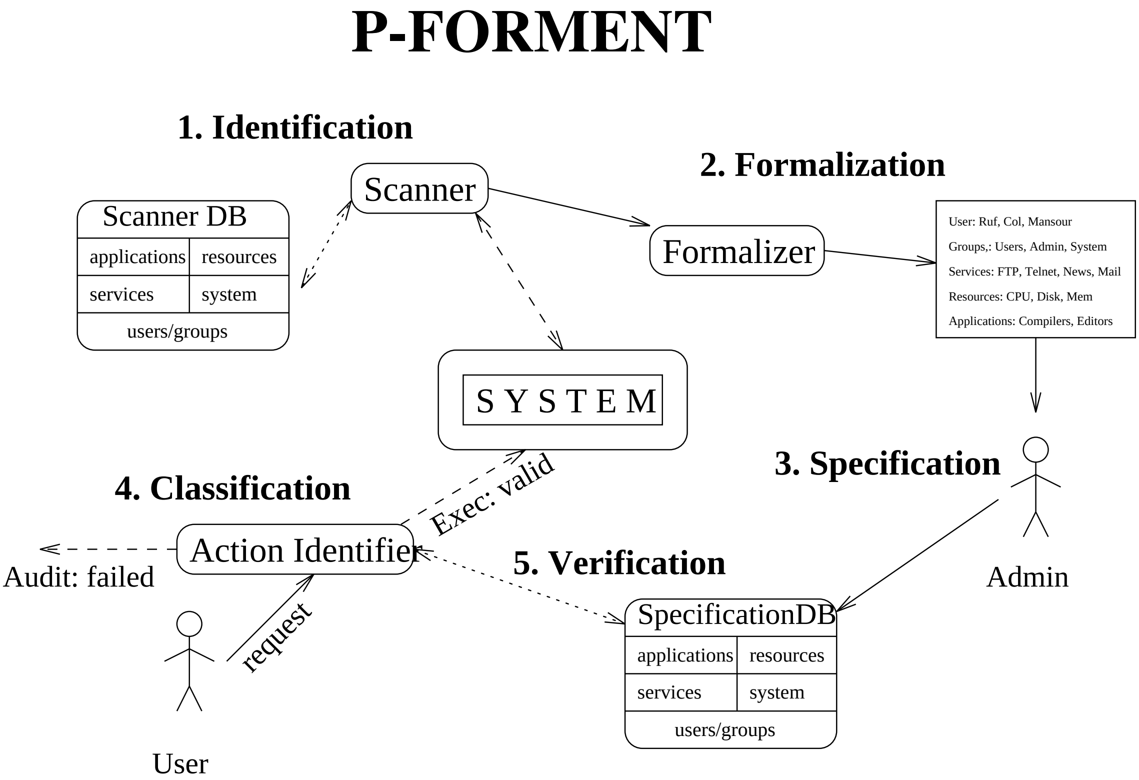

Figure 4.1: Policy Formalization and Enforcement Tool

The following is a brief description of each of the components of the system:

1. **Identification Process** - A system scanner that attempts to identify services, applications on the system. Using signature files and file identification functions the system scanner attempts to scan the system and identify each particular file. Configuration files may be specified to determine the state of the system (in much the same fashion as using COPS to identify security problems, essentially we trying to determine the configuration of the system). Any unidentified objects rely on the administrator for identification.

2. **Formalization Process** - Takes all the system objects and groups related objects (ie. Application classes - compilers, editors; Services; Users/Groups) and builds an output form.

3. **Policy Specification** - Using the form that identifies sets of services, the desired settings can be discussed and agreed upon.

   The system administrator then enters into the system the policy requirements, using a graphical tool to insert the required system settings for each system resource. Such settings may be stored in a rule-based system or database.

4. **Classification Process** - When a user accesses a service through some signature action (such as a particular system call, or executed program), the service will be identified and the user verified for access restrictions.

5. **Verification Process** - Now the system can use such a specification database to verify services to users. Additionally each individual action by the user has to be classified, to the type of service, application. This is essentially a type of program registration. Each service requires signature actions to be registered (which would be performed by the identification process) so that access can be verified.

An example scenario:
Ftp is a restricted service (typically there is a program wrapper, the application itself, or access control access to the program). The administrator may modify the ftp binary, syslog'ing all access to ftp. So how does this deter a user that has their own copy of ftp. In our system the ftp binary may be registered to a signature action, which would then discriminate from the users own copy. Once the signature for ftp is identified the user requesting the service can be verified.
Such a tool would be valuable component in an intrusion detection system. As such a stand–alone tool is a more intermediary step towards intrusion detection system and enforcing policy, and might be more useful than a full-blown intrusion detection system.

4.3 Universal Intrusion Detection Systems

Universal Intrusion Detection Systems should be developed by standardization of components of intrusion detection systems. Developing an intrusion detection framework, standardizing audit, decision and interface components would involve specifying component data structures, record formats, data management functions, data transport protocols, and language syntax.
The idea being that by developing system independent interface, a developer creating a new statistical decision engine (SDE) would assume a standard audit subsystem in which audit can be retrieved via specified functions. By utilizing the specified SDE profile data structures, a statistical engine can be developed. Conforming by outputting anomaly records and SDE status details in a standardized format.
Such components that could possibly be standardized include:

Audit Subsystem:

- audit record formats.
- audit events and event classes types.
- audit collection (pool) mechanism.
- filtering rules and syntax.
- raw audit to system independent transformation rules and syntax.
- audit transport protocol (with cryptography).
- buffering and archiving mechanisms.

Ideally would hope to develop an Audit API to supply an interface for audit control and collection allowing dynamic modification to audit masks on user/service/resource/program basis.

Processor:

- profile structures (subject/object status, data summary statistics, data values).
- penetration or misuse rule syntax.
- knowledge base structures and specification syntax("known" intrusion signatures, local "policy" specification).
- anomaly record formats (processor output).
- prevention and countermeasure protocols.

Interface:
The interface component would require less attention as GUI standards already exist (eg. X) and there is little intrusion detection specific requirements. The interface would typically consist of visual

display components: graphs , text listing (audit records and event messages), alerts, status boxes (values and flags). In addition would also consist of the audit system control interface and processor editors and configuration programs. These allow configuration and control over the standard structures in both the audit and processor systems. These would be developed with the particular components implementations, but if the standards are flexible enough it should be possible to reconfigure such programs for use in other systems.
The benefits of a general intrusion detection framework. Which is independent of any specific system requirements would allow:

- Plug in intrusion components. Allowing new audit systems, decision components, system interfaces could be developed independent of the other subsystems.
- Components such as system specific penetration signatures can be developed and released in standard format (ie. by CERT). With a standard interface specified, the component can be simply integrated into any intrusion detection system.
- Reduce replication in developments, reducing the efforts of developers to concentrate on system specific details.
- Wider acceptance of intrusion detection systems, standardized components can be ported to various platform architectures and environments.
- Community at large would participate in development of standard requirements. Such that widespread participation would result in better design and flexibility of such systems.
- Larger testing and evaluation base.

4.3.1 Universal Intrusion Audit Subsystem

The initial focus on the design of universal intrusion detection systems begins with developing an audit subsystem that is both specific to the needs of intrusion detection that separates the system dependent features of the audit subsystem from system independent ones. As previously mentioned many of the audit systems currently available produce data that is inadequate for intrusion detection. Such that often it is not possible to get data we require, such as command line options. Also many audit systems are “all or nothing” approaches. Where there is very limited control over fine tuning which events are required for auditing. The greatest difficulty is what I refer to as the fire-hose problem. Audit systems produce large volumes of data (in which most of it is irrelevant) therefore requiring lots of resources just to isolate the few drops of interesting data.
There appears to be little effort in the community to attempt solve such problems (though it has often been expressed and acknowledged that such attempts would be very beneficial). While there has been some effort in the development of intrusion detection frameworks such as in [38] and current POSIX attempts to standardize audit formats. There is still no set of standards which designers can conform. I have been investigating some of the requirements for such an approach. Focussing initially on the intrusion detection audit subsystem. I expect that a audit API could be developed, incorporating

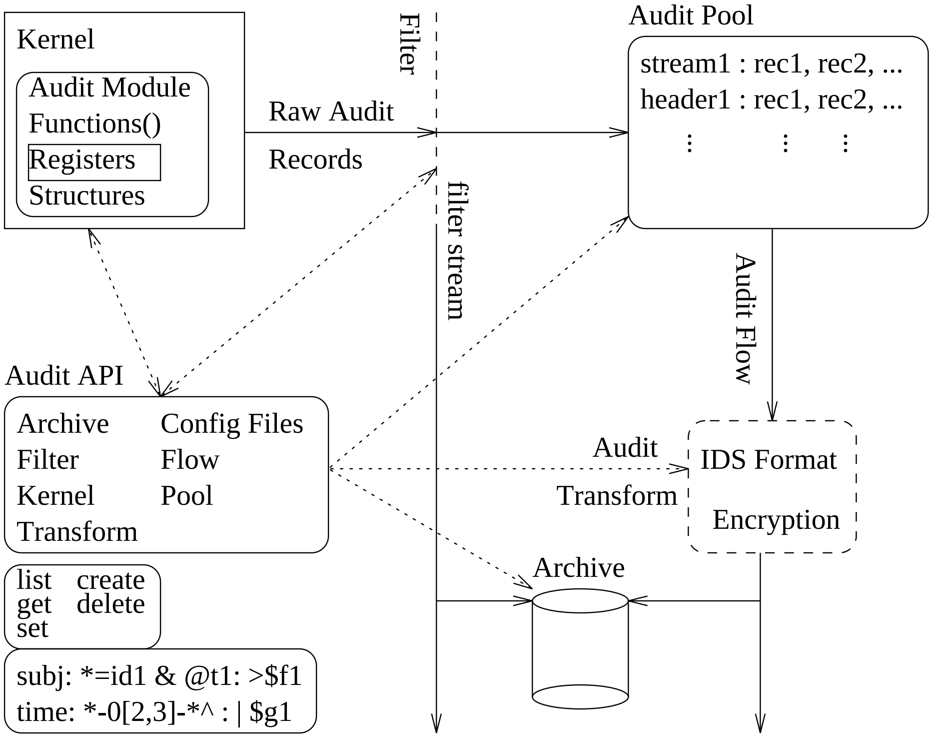

Figure 4.2: Prototype Audit Subsystem

some the previously outlined principals. Some of the design features of the audit system (see Figure 4.2) that are important to an intrusion detection audit system (in addition to permitting standardization) include:

- *Standard audit events* - high level audit events and event classes may be specified and the audit subsystem automatically converts these system independent events to the system dependent system calls.

  To demonstrate this idea we can specify (standardize) system independent audit events in which the decision or interface component can use API calls to enable the auditing of particular events (ie. audit_set(work-area,1)). Now the API calls map these requests, enabling the auditing of all the system dependent kernel system calls that relate to the high level event (ie. calls that access different file paths). When the raw (system dependent) audit data is produced from the kernel it is mapped back to the system independent audit event by the transform component. This event is specified in a standard audit record format (the result of the transform component).

  The effect is that the API remains standard with modifications only in individual function calls.

- *Dynamic audit event selection* - data structures (audit profiles) maintained in the kernel allow the audit output to be dynamically controlled. Essentially the kernel contains an audit module consisting of audit control structures (audit registers - such as event bit masks), and the necessary audit_functions() that apply the checks on what audit to produce from the kernel.

> External controls would be managed via an audit daemon (system level process), which would manage requests to the kernel for audit status (using the Audit API; eg get, lists), updates (create, delete, and modify). I expect that the Audit Module can be achieved by using kernel loadable modules, so that minimal kernel modifications are required.
>
> Therefore allowing intrusion detection systems to modify the run–time requirements of the audit system. For instance, when analyzing the audit trail particular audit events might trigger increased levels of auditing, or decreasing the level of auditing on a users who's behavior appears to be normal.

I also plan to develop prototype audit system on Linux a freely available PC based unix system. As source code for kernel and system software is available, it is possible to modify all levels of the system (however the idea is not to).

Chapter 5

Conclusion

This work outlined the issues involved with providing effective computer security, identifying many of the common problems. A lot of time was spent investigating historical intrusion incidents. To gain an understanding of the type and nature of previous incidents. Unfortunately it was not possible to present a detailed account of these incidents, nor any detailed analysis simply due to the limits on this work.

Many of the security mechanisms for Internet environments (typically UNIX/Sun) and vulnerabilities that have been exploited were examined in this work. Again, due the large number of incidents and the large amount of technical understanding required, they could not be presented in any depth. However by giving an overview of these vulnerabilities it is expected that a general understanding could be gained.

Attention was given to the historical work in intrusion detection system development. An outline of the numerous system developments was presented. And a presentation of a few of the systems was given so that it would be possible to examining how these systems attempt to assist in detecting computer system misuse. Specifically, how audit data can be utilized to develop a user activity (or behavior) profile, so that masqueraders or misfeasors can be detected. Additionally the methods used to analyze the audit data to detect events that are suspected of resembling intrusions was shown.

Furthermore, these IDS were examined in more detail by looking at the various components. And some possible extensions were presented, which were intended for future research topics.

Throughout the thesis many additional points were raised, often limited to either very little or no discussion. Specifically in the areas of legality in regards to system monitoring and user privacy issues. Also the problems regarding the development of detailed intrusion signature databases. Which is an important issue if such information are to be utilized in intrusion knowledge bases as it can be argued that such knowledge is counteractive towards other less secure systems. As it provides a source of detailed intrusion techniques for prospective intruders.

While it was difficult to present a detailed analysis of every single issue related to computer system security and the development and future extensions to intrusion detection systems. It was intended that this work provide a introductory overview of the many complex issues surrounding these systems.

Hopefully, a basic understanding of the system security mechanisms and there insecurities can reached. In which intrusion detection systems can be seen provide a potentially beneficial tool in defense against system intrusions.

Bibliography

[1] Screening External Access Link (SEAL) Introductory Guide.

[2] TIS Firewall Toolkit: Configuration and Administration, February 1994. ftp distribution.

[3] TIS Firewall Toolkit: Overview, February 1994. ftp distribution.

[4] TIS Firewall Toolkit: Users Overview, February 1994. ftp distribution.

[5] Marshall D. Abrams and Ingrid M. Olson. Rule-Based Trusted Access Control. In G.G Gable and W. J Caelli, editors, *IT Security: The Need for International Cooperation*, pages 409–420. Elsevier Science Publishers, 1992. IFIP.

[6] SERT Advisory. UNIX Computer Security Checklist. *ftp distribution*, July 1993. SA-93.06.

[7] Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalali, Ann Tamaru, and Mabry Tyson. Next Generation Intrusion Detection Expert System (NIDES): User Manual for Security Officer User Interface (SOUI) Version 1 - Alpha Release. User Manual Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, March 1993.

[8] Debra Anderson, Teresa F. Lunt, Harold Javitz, Ann Tamaru, and Alfonso Valdes. SAFEGUARD FINAL REPORT: Detecting Unusual Program Behavior Using the NIDES Statistical Component. SRI Project 2596 Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, December 1993.

[9] J P. Anderson. Computer Security Threat Monitoring and Surveillance. James P. Anderson Co, April 1980. Fort Washington PA.

[10] Frederick M. Avolio and Marcus J. Ranum. A Network Perimeter with Secure External Access.

[11] Eric Bach, Steve Bellovin, Dan Bernstein, Nelson Bolyard, Carl Ellison, Jim Gillogly, Mike Gleason, Doug Gwyn, Luke O'Connor, Tony Patti, and William Setzer. Cryptography FAQ. *Usenet sci.crypt*, August 1993. v1.0.

[12] Eugen Mate Bacic. Computer Viruses. revised version, December 1988.

[13] Eugen Mate Bacic. Process Execution Controls as a Mechanism to Ensure Consistency. In *Fifth Anual Computer Security Applications Conference*, pages 114–120, 1989.

[14] Eugen Mate Bacic. Crying "Wolf". In *Proceedings of the Second Annual Canadian Computer Security Symposium*, pages 237–245, Ottawa, Canada, 1990.

[15] Eugen Mate Bacic. Process Execution Control: Revisited. revised, 1990.

[16] Eugen Mate Bacic. Viruses: Steaming the Tide, 1991.

[17] Eugen Mate Bacic. The Canadian Criteria, Version 3.0 & The US Federal Criteria, Version 1.0. CC v3.0, FC v1.0, 1993.

[18] Bob Bales. Internet Firewalls Frequently Asked Questions. *Usenet comp.security.unix FAQ*, November 1994.

[19] Fuat Baran, Howard Kaye, and Margarita Suarez. Security Breaches: Five Recent Incidents at Columbia University. ftp distribution.

[20] Tony Bartoletti. *Security Profile Inspector (SPI) for the UNIX Operating System: User Guide version 2.1*. Lawrence Livermore National Laboratory, dids-92-16 edition, July 1992.

[21] David S. Bauer and Michael E. Koblentz. NIDX - An Expert System for Real-Time Network Intrusion Detection. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 98–106. IEEE, April 1988. 11-13th, Washington, DC.

[22] S. M. Bellovin. Security Problems in the TCP/IP Protocol Suite. *Computer Communication Review*, 19(2):32–48, April 1989.

[23] Steven Bellovin. There Be Dragons. In *Proceedings of the Third Usenix UNIX Security Symposium*, September 1992.

[24] Steven M. Bellovin and Michael Merritt. Limitations of the Kerberos Authentication System. In *Usenix Winter 91*, Dallas, TX, 1991.

[25] Matt Bishop. An Overview of Computer Viruses in a Research Environment. Technical report, Dartmouth College, 1991. PCS-TR-91-156.

[26] Paul Boedges. Air Force mounts offensive against computer crime. In *Government Computer News* [144], page 51. 12-16th, Orlando, FL.

[27] Kevin Brady. Integrating B2 Security Into a UNIX System. In G.G Gable and W. J Caelli, editors, *IT Security: The Need for International Cooperation*. Elsevier Science Publishers, 1992. IFIP.

[28] Russell L. Brand. Coping with the Threat of Computer Security Incidents - A Primer from Prevention through Recovery, June 1990.

[29] Sheila L. Brand. Department of Defense Trusted Computer System Evaluation Criteria. National Computer Security Center, Fort Meade, Maryland, December 1985. DOD 5200.28-STD.

[30] Ralf Burger. *Computer Viruses: A high-tech disease.* Abacus. Data Becker GmbH, 1988. ISBN 1-55755-043-3.

[31] F. Carrettoni, S. Castano, G. Martella, and P. Samarati. RETISS: A Real Time Security System For Threat Detection Using Fuzzy Logic. In *Proceedings 25th Anuual IEEE International Carnahan Conference on Security Technology*, October 1991. 1-3rd, Taipei, Taiwan.

[32] Leslie S. Chalmers. An analysis of the differences between the computer security practices in the military and private sectors. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 71–4, April 1986. 7-9th, Oakland, California.

[33] Bill Cheswick. An Evening with Berferd: In Which a Cracker is Lured, Endured and Studied. Proceedings Winter USENIX Conference, January 1992.

[34] William R. Cheswick and Steven M. Bellovin. *Firewalls and Internet Security.* Professional Computing Series. Addison-Wesley Publishing Company, 1994. ISBN 0-201-63357-4.

[35] David A Curry. Improving the Security of your UNIX System. Technical report, SRI International, April 1990. ITSTD-721-FR-90-21.

[36] Hervé Debar, Monique Becker, and Didier Siboni. A Neural Network Component for an Intrusion Detection System. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 240–250. IEEE and IACR, May 1992. 4-6th, Oakland, CA.

[37] Hervé Debar and Bernadette Dorizzi. An Application of a Recurrent Network to an Intrusion Detection System. In *IJCNN International Joint Conference in Neural Networks*, pages 478–483 vol 2. IEEE and Int'l Neural Network Society, June 1992. 7-11th, Baltimore, MD.

[38] Dorothy E. Denning. An Intrusion-Detection Model. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 1986.

[39] Peter J. Denning. *Computers Under Attack: Intruders, Worms and Viruses.* ACM Press, 1990. ISBN 0-201-53067-8.

[40] Yves Deswarte, Laurent Blain, and Jean-Charles Fabre. Intrusion Tolerance in Distributed Computer Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 110–21, May 1991. 20-22nd Oakland, CA.

[41] Yves Deswarte, Jean-Charles Fabre, Jean-Michel Fray, David Powell, and Pierre-Guy Ranéa. SATURNE: A distributed computing system which tolerates faults and intrusions. In *Proceedings workshop on the future trends of distributed computer systems in the 1990's*, pages 329–38, September 1988. 14-16th Hong Kong.

[42] Deborah D. Downs, Jerzy R. Rub, Kenneth C. Kung, and Carole S. Jordan. Issues In Discretionary Access Control. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 208–18, April 1985. 22-24th Oakland, California.

[43] Mark W. Eichin and Jon A. Rochlis. With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988. In *Proceedings IEEE Symposium on Research in Security and Privacy*, 1989. ftp distribution.

[44] Hackers et al. Hacker Jargon File. *Project Gutenberg ETEXT*, July 1992.

[45] Taran King et. al. Phrack Philes. CUD ETexts Archives, 1985 - 1994. Volume 1 - 45.

[46] Paul Fahn. FAQ About Today's Cryptography. *Usenet sci.crypt*, September 1993. v2.0.

[47] Dan Farmer and Wietse Venema. Improving the Security of Your Site by Breaking Into it. *Usenet comp.security*, 1993.

[48] Daniel Farmer and Eugene H. Spafford. The COPS Security Checker System. Technical report, Purdue University, September 1991. Technical Report CSD-TR-993.

[49] Jeremy Frank. Artifical Intelligence and Intrusion Detection: Current and Future Directions, June 1994. ftp distribution.

[50] Jean-Michel Fray, Yves Deswarte, and David Powell. Intrusion-Tolerance Using Fine-Grain Fragmentation-Scattering. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 194–201, 1986.

[51] Simson Garfinkel. Security Article Extracts: Legalities. ftp distribution, 1987.

[52] Simson Garfinkel and Gene Spafford. *Practical Unix Security*. Nutshell Handbook Series. O'Reilly & Associates Inc, August 1993. ISBN 0-937175-72-2.

[53] Thomas D. Garvey and Teresa F. Lunt. Model-Based Intrusion Detection. In *Proceedings of the Fourteenth National Computer Security Conference*, October 1991.

[54] Thomas D. Garvey and Teresa F. Lunt. Using Models of Intrusions. In *Third Workshop on Computer Security Incidence Handling*, August 1991.

[55] Naji Habra, Baudouin Le Charlier, and Abdelaziz Mounji. Advanced Security Audit Trail Analysis on uniX - Implementation design of the NADF Evaluator. Technical report, Institut DÍnformatique FUNDP, September 1994.

[56] Naji Habra, Baudouin Le Charlier, and Abdelaziz Mounji. ASAX: Specification of some extensions to the Analyzer. Technical report, Institut DÍnformatique FUNDP, September 1994.

[57] Naji Habra, Baudouin Le Charlier, Abdelaziz Mounji, and Isabelle Mathien. Preliminary report on Advanced Security Audit Trail Analysis on uniX. Technical report, Institut DÍnformatique FUNDP, September 1994.

[58] Naji Habra and Isabelle Mathien. ASAX: Software Architecture and Rule–Based Language for Universal Audit Trail Analysis. In *Proceedings of ESORICS 92 European Symposium on Research in Computer Security*, November 1992. 23-25 Toulouse.

[59] Katie Hafner and John Markoff. *Cyberpunk*. Fourth Estate Limited, 1991. ISBN 1-872180-94-9.

[60] Neil M. Haller. The S/KEY One–Time Password System.

[61] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A Network Security Monitor. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 296–304. IEEE, May 1990. 7-9th, Oakland, CA.

[62] Paul Helman and Gunar Liepins. Statistical Foundations of Audit Trail Analysis for the Detection of Computer Misuse. In *IEEE Transactions on Software Engineering*, September 1993. vol 19, no 9.

[63] Paul Helman, Gunar Liepins, and Wynette Richards. Foundations of Intrusion Detection. In *Proceedings of Computer Security Foundations Workshop V*, pages 114–120. IEEE, June 1992. 16-18th, Franconic, NH.

[64] I.S Herschberg. Make the Tigers Hunt for You. *Computers & Security*, pages 197–203, 1988.

[65] David K. Hess, David R. Safford, and Udo W. Pooch. A Unix Network Protocol Security Study: Network Information Service. ftp distribution.

[66] Lance J. Hoffman and Russell J. Davis. Security Pipeline Interface (SPI). In *Proceedings 6th Annual Computer Security Applications Conference*, pages 349–55, December 1990. 3-7th Tucson, AZ.

[67] Don Holden. A Rule-Based Intrusion Detection System. In G.G Gable and W. J Caelli, editors, *IT Security: The Need for International Cooperation*, pages 433–440. Elsevier Science Publishers, 1992. IFIP.

[68] Craig Hunt. *TCP/IP Network Administration*. Nutshell Handbook Series. O'Reilly & Associates Inc, September 1993. ISBN 0-937175-82-X.

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

[70] 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.

[71] John Ioannidis and Matt Blaze. The Architecture and Implementation of Network–Layer Security Under Unix. ftp distribution.

[72] R. Jagannathan, Teresa Lunt, Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalali, Hal Javitz, Peter Neumann, Ann Tamaru, and Alfonso Valdes. System Design Document:

Next-Generation Intrusion Detection Expert Sytem (NIDES). System Design Document; A007, A008, A009, A011, A012, A014 Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, March 1993.

[73] R. Jagannathan, Teresa Lunt, Fred Gilham, Ann Tamaru, Caveh Jalali, Peter Neumann, Debra Anderson, Thomas Garvey, and John Lowrance. Requirements Specification: Next-Generation Intrusion Detection Expert System (NIDES). Requirements Specifications A001, A002, A003, A004, A006 Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, September 1992.

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

[75] Harold S. Javitz and Alfonso Valdes. The NIDES Statistical Component: Description and Justification. A010-Justification Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, March 1994.

[76] Harold S. Javitz, Alfonso Valdes, Teresa Lunt, Ann Tamaru, Mabry Tyson, and John Lowrance. Next Generation Intrusion Detection Expert System (NIDES) 1. Statistical Algorithms Rationale 2. Rationale for Proposed Resolver. A016- Rationales Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, March 1993.

[77] Laurent Jolia-Ferrier. Penetration Testing As An Audit Tool. In *Fourth IFIP International Conference on Computer Security*, pages 119–25, December 1986. 2-4th Monte Carlo, Monaco.

[78] Paul A. Karger. Limiting the Damage Potential of Discretionary Trojan Horses. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 32–7, April 1987. 27-29th Oakland, California.

[79] Jeffrey O. Kephart and Steve R. White. Measuring and Modeling Computer Virus Prevalence. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 2–15, May 1993.

[80] Susan Kerr. Using AI to Improve Security. *DATAMATION*, pages 57–60, February 1990.

[81] Gene H. Kim and Eugene H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. Technical report, Purdue University COAST Laboratory, November 1993. Technical Report CSD-TR-93-071.

[82] Gene H. Kim and Eugene H. Spafford. Experiences with Tripwire: Using Integrity Checkers for Intrusion Detection. Technical report, Purdue University COAST Laboratory, February 1994. Technical Report CSD-TR-93-071.

[83] Christopher Klaus. What if your Machines are Compromised by an Intruder. *Usenet comp.security FAQ*, September 1994. v1.7.

[84] Henry M. Kluepfel. In Search of the Cuckoo's Nest. In *Proceedings 25th Annual International Carnahan Conference on Security Technology*, pages 181–91, October 1991. 1-3rd Taipei, Taiwan.

[85] Calvin Ko, Deborah A. Frincke, Terrence Goan Jr., L. Todd Heberlein, Karl Levitt, Biswanath Mukherjee, and Christopher Wee. Analysis of an algorithm for distributed recognition and accountability.

[86] Sandeep Kumar and Eugene H. Spafford. An Application of Pattern Matching in Intrusion Detection. Technical report, COAST Project Purdue University, June 1994. CSD-TR-94-013.

[87] Bill Landreth. *Out of the Inner Circle.* Tempus Books of Microsoft Press, 1989. ISBN 1-55615-223-X.

[88] Linda Lankewicz and Mark Benard. Real–Time Anomaly Detection Using A Nonparametric Pattern Recognition Approach. In *Proceedings of Seventh Annual Computer Security Conference*, December 1991. 2-6th, San Antonio, TX.

[89] Steven Levy. *Hackers: Heroes of the Computer Revolution.* Anchor Press/Doubleday, 1984.

[90] G. E. Liepins and H. S. Vaccaro. Intrusion Detection: Its Role and Validation. In *Computers and Security*, pages 347–355. Elsevier Science Publishers Ltd, 1992.

[91] Justin J. Lister. Enforcing Computer Policy. In *14th Intrusion Detection Workshop*, Baltimore MD, USA, October 1994. 17th National Computer Security Conference.

[92] Mike Loukides. *System Performance Tuning.* Nutshell Handbook Series. O'Reilly & Associates Inc, December 1992. ISBN 0-937175-60-9.

[93] Teresa F. Lunt. Automated Audit Trail Analysis and Intrusion Detection: A Survey. In *Proceedings of the 11th National Computer Security Conference*, October 1988.

[94] Teresa F. Lunt. Detecting Intruders in Computer Systems. In *Proceedings of the Sixth Anual Symposium and Technical Displays on Physical and Electronic Security*, 1989.

[95] Teresa F. Lunt. Real-Time Intrusion Detection. In *COMPCON Spring Proceedings*, 1989.

[96] Teresa F. Lunt. IDES: An Intelligent System fo Detecting Intruders. In *Proceedings of the Symposium Computer Security, Threat and Countermeasures*, November 1990.

[97] Teresa F. Lunt. Using Statistics to Track Intruders. In *Proceedings of the Joint Statistical Meetings of the American Statistical Association*, August 1990.

[98] Teresa F. Lunt. A survey of intrusion detection techniques. *Computers and Security*, 12:405–418, 1993.

[99] Teresa F. Lunt. Detecting Intruders in Computer Systems. In *Proceedings of the Conference on Auditing and Computer Technology*, 1993.

[100] Teresa F. Lunt and Debra Anderson. Software Requirements Specification: Next–Generation Intrusion Detection Expert System. Technical report, SRI International, March 1993.

[101] Teresa F. Lunt and R. Jagannathan. A Prototype Real-Time Intrusion-Detection Expert System. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, April 1988.

[102] Teresa F. Lunt, R. Jagannathan, Rosanna Lee, and Alan Whitehurst. Knowledge-Based Intrusion Detection. In *Proceedings of the AI Systems in Government Conference*, March 1989.

[103] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Caveh Jalali, Peter G. Neumann, Harold S. Javitz, Alfonso Valdes, and Thomas D. Garvey. A Real-Time Intrusion-Detection Expert System (IDES). Final Technical Report Tel: (415) 326-6200, Fax: 326-5512, Telex: 334486, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, February 1992.

[104] Teresa F. Lunt, Ann Tamaru, Fred Gilham, R. Jagannathan, Peter G. Neumann, and Caveh Jalali. IDES: A Progress Report. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, September 1990.

[105] Victor H. Marshall. Intrusion Detection in Computers. Technical report, Trusted Information Systems (TIS), January 1991.

[106] Noelle McAuliffe, Dawn Wolcott, Lorrayne Schaefer, Nancy Kelem, Brian Hubbard, and Theresa Haley. Is Your Computer Being Misused ? A Survey of Current Intrusion Detection System Technology. In *Proceedings of the Sixth Annual Computer Security Applications Conference*, pages 260–272, 1990.

[107] Gordon R. Meyer. The Social Organization of the Computer Underground. Master's thesis, Northern Illinois University, August 1989.

[108] Jeffery C. Mogul. Simple and Flexible Datagram Access Controls for Unix–based Gateways. Technical report, digital WRL, 1989.

[109] Refik Molva, Gene Tsudik, Els Van Herreweghen, and Stefano Zatti. KryptoKnight Authentication and Key Distribution System, 1993.

[110] Robert Morris and Ken Thompson. Password Security: A Case History. UNIX Programmers Manual or Systems Manager Manual. ftp distribution.

[111] Robert T. Morris. A Weakness in the 4.2BSD Unix TCP/IP Software, February 1985. ftp distribution.

[112] Abdelaziz Mounji, Baudouin Le Charlier, Denis Zampunieŕis, and Naji Habra. Preliminary report on Distributed ASAX. Technical report, Institut DÍnformatique FUNDP, May 1994.

[113] Alec Muffett. Almost Everything You Ever Wanted To Know About Security. *Usenet comp.security FAQ*, December 1993. v2.2.

[114] Micheal Neuman and Gary Christoph. The Operator Shell: A Means of Privilege Distribution. ftp.c3.lan.gov:/pub/mcn/osh.tar.Z.

[115] William Michael Newberry. *Active Intruder Detection: Some Aspects of Computer Security and User Authentication*. PhD thesis, Australian Defence Force Academy University College University of New South Wales, February 1991.

[116] M. S. Obaidat. A methodology for improving computer access security. In *Computers and Security*, pages 657–662. Elsevier Science Publishers Ltd, 1993.

[117] Eiji Okamoto. Proposal for Integrated Security Systems. In *Proceedings 2nd International Conference on Systems Integration*, pages 354–8, June 1992. 15-18th Morristown, New Jersey.

[118] Karen L. Petersen. IDA - Intrusion Detection Alert. In *Proceedings of the Sixtenth Annual International Computer Software and Applications Conference*, pages 306–311. IEEE, September 1992. 21-25th, Chicago, IL.

[119] Richard D. Pethia and Kenneth R. van Wyk. Computer Emergency Response - An International Problem. cert.sei.cmu.edu, 1990.

[120] John A. Pew. *Guide to Solaris*. SunSoft Offical Guide. Ziff-Davis Press, 1993. ISBN 1-56276-087-4.

[121] Charles P. Pfleeger. *Security In Computing*. Student. Prentice-Hall Inc, 1989. ISBN 0-13-799016-2.

[122] Charles P. Pfleeger, Shari Lawrence Pfleeger, and Mary Frances Theofanos. A Methodology For Penetration Testing. *Computers & Security*, pages 613–20, 1989.

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

[124] Phillip A. Porras and Richard A. Kemmerer. Penetration State Transition Analysis A Rule-Based Intrusion Detection Approach. In *Proceedings of the Eigth Annual Computer Security Applications Conference*, pages 220–229. Aerospace Computer Security Associates and IEEE and ACM SIGSAC, November 1992. 30-4th, San Antonio, TX.

[125] Phillip Andrew Porras. STAT A State Transistion Analysis Tool For Intrusion Detection. Master's thesis, University of California Santa Barbara, 1992.

[126] Stephen A. Rago. *UNIX System V Network Programming*. Professional Computing Series. Addison-Wesley Publishing Company, June 1993. ISBN 0-201-56318-5.

[127] Marcus J. Ranum and Frederick M. Avolio. A Toolkit and Methods for Internet Firewalls.

[128] Raptor Systems Incorporated. *Eagle Network Security Management System*, version 2.2 edition, 1992. ftp distribution.

[129] Robert B. Reinhardt. An Architectural Overview of UNIX Network Security. *Usenet alt.security*, November 1993. V4.

[130] Lyford D. Rich and Chyan Yang. Network Security in the UNIX Environment. In *Milcom '92 Communications Command, Control and Intelligence Conference*, pages 1101–5, October 1992. 11-14th San Diego, CA.

[131] Dennis M. Ritchie. On the Security of UNIX. UNIX Programmers Manual or Systems Manager Manual, June 1977. ftp distribution.

[132] David R. Safford, David K. Hess, and Douglas Lee Schales. Secure RPC Authentication (SRA) for TELNET and FTP. In *Proceedings of the Fourth USENIX Security Symposium*, 1993.

[133] David R. Safford, Douglass Lee Schales, and David K. Hess. The TAMU Security Package: An Ongoing Response to Internet Intruders in an Academic Environment. USENIX Assocation, 1993. Proceedings of the Fourth USENIX Security Symposium.

[134] Samuel I. Schaen and Brian W. McKenny. Network Auditing: Issues and Recommendations. In *Proceedings 7th Annual Computer Security Applications Conference*, pages 66–79, December 1991. 2-6th San Antonio, TX.

[135] Bruce Schneier. *Applied Cryptography.* John Wiley & Sons, 1993. ISBN 0-471-59756-2.

[136] Christoph L. Schuba and Eugene H. Spafford. Countering Abuse of Name–Base Authentication. ftp distribution.

[137] Jennifer Seberry and Josef Pieprzyk. *Cryptography: An Introduction to Computer Security.* Advances in Computer Science Series. Prentice Hall, 1989. ISBN 0-13-194986-1.

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

[139] Donn Seeley. A Tour of the Worm. ftp distribution.

[140] 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*, May 1990.

[141] Shiuh-Pyng W. Shieh and Virgil D. Gligor. Auditing the Use of Covert Storage Channels in Secure Systems. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 285–95, May 1990. 7-9th, Oakland, California.

[142] Shiuhpyng W. Shieh and Virgil D. Gligor. A Pattern-Oriented Intrusion-Detection Model and Its Applications. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 327–342. IEEE and IACR, May 1991. 20-22nd, Oakland, CA.

[143] A. Silberschatz, J. Peterson, and P. Galvin. *Operating System Concepts.* Student Computing Series. Addison-Wesley Publishing Company, 1991. ISBN 0-201-54873-9.

[144] Stephen E. Smaha. Haystack: An Intrusion Detection System. In *Proceedings of the Fourth Aerospace Computer Security Applications Conference*, pages 37–44. IEEE and ASIS, December 1988. 12-16th, Orlando, FL.

[145] Steven 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 *COMPCON Spring Proceedings*, pages 170–176. IEEE, February 1991. 25-1st, San Francisco, CA.

[146] Eugene H. Spafford. The Internet Worm: An Analysis. Technical report, Purdue University, December 1988. Technical Report CSD-TR-823.

[147] Eugene H. Spafford. Some Musings on Ethics and Computer Break–Ins, 1989.

[148] Eugene H. Spafford. Are Computer Hacker Break–ins Ethical ? Technical report, Purdue University, April 1991. Technical Report CSD-TR-994.

[149] Eugene H. Spafford. OPUS: Preventing Weak Password Choices. Technical report, Purdue University, June 1991. Technical Report CSD-TR-92-028.

[150] Eugene H. Spafford. The Internet Worm Incident. Technical report, Purdue University, September 1991. Techinical Report CSD-TR-933.

[151] Eugene H. Spafford. Observing Reusable Password Choices. Technical report, Purdue University, July 1992. Technical Report CSD-TR-92-049.

[152] Paul Sprirakis, Sokratis Katsikas, Dimitris Gritzalis, Francois Allegre, Dimitris Androutsopouls, John Darzentas, Claude Gigante, Dimitris Karagiannis, Heikki Putkonen, and Thomas Spyrou. SecureNet: A Network-Oriented Intelligent Intrusion Prevention And Detection System. In *IFIP Security 94*, pages 2–16, 1994.

[153] Charles Spurgeon. UTnet Guide to UNIX System Security, 1990. ftp distribution.

[154] Site Security Policy Handbook Working Group (SSPHWG). Site Security Handbook. Requests For Comments (RFC'S), July 1991. RFC-1244.

[155] William Stallings. *Networking Standards: A Guide to OSI, ISDN, LAN and MAN Standards.* Addison-Wesley Publishing Company, January 1993. ISBN 0-201-56357-6.

[156] Bruce Sterling. *The Hacker Crackdown.* Penguin Books, 1992.

[157] Hal Stern. *Managing NFS and NIS.* Nutshell Handbook Series. O'Reilly & Associates Inc, April 1992. ISBN 0-937175-75-7.

[158] W. Richard Stevens. *Advanced Programming in the UNIX Environment.* Professional Computing Series. Addison-Wesley Publishing Company, June 1992. ISBN 0-201-56317-7.

[159] Cliff Stoll. Stalking the wily hacker. *Communications of the ACM*, 31(5):484–98, May 1988.

[160] Cliff Stoll. Stalking The Wily Hacker - An Update, 1993.

[161] Clifford Stoll. *The Cuckoo's Egg.* Pan Books Ltd, 1991. ISBN 0-330-31742-3.

[162] SunSoft. *Solaris 2.3 Administering Security, Performance and Accounting - Part 1*, part no: 801-5282-10 edition, October 1993.

[163] SunSoft, Part No: 801-5282-10. *Solaris 2.3 Administering Security, Performance and Accounting - Part 2*, October 1993.

[164] SunSoft. *Solaris SHIELD Basic Security Module*, part no: 801-5285-10 edition, October 1993.

[165] Andrew S. Tanenbaum. *Computer Networks.* International Editions. Prentice Hall, 1989. ISBN 0-13-166836-6.

[166] CS Telecom. Hyperview: Product Description, September 1994.

[167] William T. Tener. Discovery: An Expert System in the Commercial Data Security Environment. In A. Grissonnanche, editor, *Security and Protection in Information Systems*, pages 261–268. Elsevier Science Publishers, 1989. IFIP.

[168] Henry S. Teng, Kaihu Chen, and Stephen C-Y Lu. Adaptive Real–Time Anomaly Detection Using Inductively Generated Sequential Patterns. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, May 1990.

[169] Henry S. Teng, Kaihu Chen, and Stephen C-Y Lu. Security Audit Trail Analysis Using Inductively Generated Predictive Rules, 1990.

[170] Jim Thomas and Gordon Meyer. Computer Underground Digest. CUD ETexts Archives, 1990 - 1994. Volumes 1.00 - 5.36.

[171] G. Winfield Treese and Alec Wolman. X Through the Firewall, and Other Application Relays. Technical report, Cambridge Research Laboratory (DEC), May 1993. CRL 93/10.

[172] Gene Tsudik and Rita Summers. AudES - an Expert System for Security Auditing. In *Proceedings of AAAI Conference on Innovative Applications in Artificial Intelligence*, 1990.

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

[174] Leendert van Doorn. Computer Break-ins: A Case Study. ftp distribution.

[175] Wietse Venema. TCP WRAPPER : Network monitoring, access control, and booby traps, 1992.

[176] Will Spencer (Voyager). alt.2600/#Hack FAQ. *Usenet alt.2600*, November 1994. Beta Version: .008.

[177] Patricia W. Wade. Data Security - Can auditor involvement or use of auditing techniques help in building more secure systems ? In *Proceedings of the Communications on the Move Conference.* ICCS, ISITA, IEEE, November 1992. 6-20th, Singapore.

[178] J R. Winkler and W J. Page. Intrusion and Anomaly Detection in Trusted Systems. In *Proceedings of the Fifth Annual Computer Security Applications Conference*, pages 39–45, December 1989.

[179] Janice Winsor. *Solaris: Advanced System Administrator's Guide*. SunSoft Offical Guide. Ziff-Davis Press, 1993. ISBN 1-56276-131-5.