A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems

by

Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degrees of

Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

Author ……………………………………………………………………………………...
Department of Electrical Engineering and Computer Science,
May 21, 1999

Certified by………………………………………………………………………………
Richard Lippmann
Senior Scientist, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by……………………………………………………………………………...
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

*This work was sponsored by the Department of Defense Advanced Research Projects Agency (DARPA). Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by DARPA.

A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems

by
Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

The 1998 DARPA intrusion detection evaluation created the first standard corpus for evaluating computer intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and new attacks embedded in a large amount of normal background traffic. The corpus was collected from a simulation network that was used to automatically generate realistic traffic—including attempted attacks.

The focus of this thesis is the attacks that were developed for use in the 1998 DARPA intrusion detection evaluation. In all, over 300 attacks were included in the 9 weeks of data collected for the evaluation. These 300 attacks were drawn from 32 different attack types and 7 different attack scenarios. The attack types covered the different classes of computer attacks and included older, well-known attacks, newer attacks that have recently been released to publicly available forums, and some novel attacks developed specifically for this evaluation.

The development of a high quality corpus for evaluating intrusion detection systems required not only a variety of attack types, but also required realistic variance in the methods used by the attacker. The attacks included in the 1998 DARPA intrusion detection evaluation were developed to provide a reasonable amount of such variance in attacker methods. Some attacks occur in a single session with all actions occurring in the clear, while others are broken up into several sessions spread out over a long period of time with the attacker taking deliberate steps to minimize the chances of detection by a human administrator or an intrusion detection system. In some attacks, the attacker breaks into a computer system just for fun, while in others the attacker is interested in collecting confidential information or causing damage. In addition to providing detailed descriptions of each attack type, this thesis also describes the methods of stealthiness and the attack scenarios that were developed to provide a better simulation of realistic computer attacks.

Thesis Supervisor: Richard Lippmann
Title: Senior Scientist, MIT Lincoln Laboratory

Acknowledgments

First, I would like to thank my thesis advisor, Rich Lippmann, for keeping my thesis on track and for providing me with many valuable ideas and insights. I would also like to thank Rob Cunningham and Dave Fried for taking the time to read and comment on early drafts of this thesis. I appreciate the support of Rich, Rob, Dave, Isaac Graf, Seth Webster, Dan Weber, and Jonathon Korba, and the rest of the intrusion detection evaluation staff, who made working in this group a fun and highly educational experience. Finally, I would like to thank Debbie and my parents, whose love and support fuel all of my endeavors.

Contents

List of Figures

Chapter 1

Introduction

1.1 1998 DARPA Intrusion Detection System Evaluation

Heavy reliance on networked computer resources and the increasing connectivity of these networks has greatly increased the potential damage that can be caused by attacks launched against computers from remote sources. These attacks are difficult to prevent with firewalls, security policies, or other mechanisms because system and application software is changing at a rapid pace, and this rapid pace often leads to software that contains unknown weaknesses or bugs. Intrusion detection systems are designed to detect those attacks that inevitably occur despite security precautions. Some intrusion detection systems detect attacks in real time and can be used to stop an attack in progress. Others provide after-the-fact information about attacks that can be used to repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type [43].

Many parties are working on the development of intrusion detection systems, including universities, commercial software companies, and organizations within the Department of Defense. As these groups explore different methods and develop various new systems for intrusion detection, it is clearly advantageous to have a means of evaluating the success of these systems in detecting attacks. The best environment for testing and evaluation of an intrusion detection system is the actual environment in which it will be used. However, research groups often do not have access to operational networks on which to test their systems, and these systems (especially while they are still in early development) are tested in a simulated environment. The ability to perform accurate testing and evaluation in a simulated environment requires high-quality data that is similar to the traffic (including attacks) that one finds on operational networks. In general, this data is difficult to acquire because it contains private information and reveals potential vulnerabilities of the networks from which the data is collected. These factors led to DARPA sponsorship of MIT Lincoln Laboratory's 1998 intrusion detection evaluation, which created the first standard corpus for the evaluation of intrusion detection systems.

The 1998 intrusion detection evaluation was the first of an ongoing series of yearly evaluations conducted by MIT Lincoln Laboratory under DARPA ITO and Air Force Research Laboratory sponsorship. These evaluations contribute significantly to the intrusion detection research field by providing direction for research efforts and calibration of current technical capabilities. The 1998 evaluation was designed to be simple, to focus on core technology issues, and to encourage the widest possible participation by eliminating security and privacy concerns and by providing data types that are used by the majority of intrusion detection systems. Data for the first evaluation was made available in the summer of 1998. The evaluation itself occurred towards the end of the summer. A follow-up meeting for evaluation participants and other interested parties was held in December 1998 to discuss the results of the evaluation.

1.2 The Development of Attacks for the 1998 DARPA Evaluation

This thesis describes the computer attacks that were included in the 1998 DARPA intrusion detection evaluation. A large sample of actual computer attacks was needed to accurately test the performance of intrusion detection systems. These attacks needed to cover the different classes of attack types. Many of the attacks used in the evaluation were drawn from public sources, but some novel attacks were developed specifically for use in this evaluation. In all cases, these attacks had to be adapted to work reliably in the largely automated simulation network from which the 1998 DARPA evaluation data were collected. Later sections of this thesis discuss the methods that were developed to create realistic simulations of computer intrusion scenarios, and the methods that were developed to vary the degree of attack stealthiness. People who attack computer networks often have goals beyond simply gaining access to a system. Some attackers break into computers simply for the challenge, others are interested in collecting information, and some are motivated by the desire to cause damage. Attackers also vary in their level of sophistication, and an accurate evaluation of intrusion detection systems requires testing how well the systems are able to detect attacks from all types of attackers—from the relative novice who is not aware that an intrusion detection system is monitoring a network, to the sophisticated, experienced cracker who knows about intrusion detection systems and takes steps to avoid being caught.

1.3 Outline of the Thesis

Chapter 2 presents background information about computer attacks and intrusion detection systems. This provides perspective for later discussion on the development of attacks for use in evaluating these systems. The different types of computer attacks and some of the many intrusion detection strategies that are currently being developed are discussed.

Chapter 3 describes the simulation network that was used to collect data for the 1998 DARPA intrusion detection evaluation. This network consisted of 11 computers and one router that, with the aid of software developed for use in the evaluation, simulated a large network consisting of hundreds of machines, and thousands of users.

Chapters 4, 5, 6, 7, 8, and 9 focus on the different types of attacks that were developed for use in the evaluation. Chapter 4 presents a taxonomy of computer attacks that was useful in choosing attack types to include in the evaluation. Chapter 5 is an introduction to Chapters 6 through 9 which discuss in detail the specific attacks within each of the four broad categories of attacks included in the evaluation (denial of service attacks, attacks that give a local user superuser access, attacks that give a remote user local access, and probes).

Chapters 10 and 11 discuss the attack scenarios and stealthy methods that were used to create a more realistic simulation of actual computer attacks.

Chapter 12 describes the processes that were used to plan, collect, and verify the attack instances that were included in the 1998 DARPA intrusion detection evaluation.

Finally, in Chapter 13 the results of the 1998 DARPA intrusion detection evaluation are summarized and discussed with a focus on the implications of these results for both the intrusion detection research community, and the Lincoln Laboratory group that will be conducting future evaluations.

Chapter 2

Background

2.1 Overview of Computer Attacks

In its broadest definition, a computer attack is any malicious activity directed at a computer system or the services it provides. Examples of computer attacks are viruses, use of a system by an unauthorized individual, denial-of-service by exploitation of a bug or abuse of a feature, probing of a system to gather information, or a physical attack against computer hardware. A subset of the possible types of computer attacks were included in the 1998 DARPA intrusion detection system evaluation, including: (1) Attacks that allow an intruder to operate on a system with more privileges than are allowed by the system security policy, (2) Attacks that deny someone else access to some service that a system provides, or (3) Attempts to probe a system to find potential weaknesses.

The following paragraphs provide some examples of the many ways that an attacker can either gain access to a system or deny legitimate access by others.

Social Engineering: An attacker can gain access to a system by fooling an authorized user into providing information that can be used to break into a system. For example, an attacker can call an individual on the telephone impersonating a network administrator in an attempt to convince the individual to reveal confidential information (passwords, file names, details about security policies). Or an attacker can deliver a piece of software to a user of a system which is actually a trojan horse containing malicious code that gives the attacker system access.

Implementation Bug: Bugs in trusted programs can be exploited by an attacker to gain unauthorized access to a computer system. Specific examples of implementation bugs are buffer overflows, race conditions, and mishandled of temporary files.

Abuse of Feature: There are legitimate actions that one can perform that when taken to the extreme can lead to system failure. Examples include opening hundreds of telnet connections to a machine to fill its process table, or filling up a mail spool with junk e-mail.

System Misconfiguration: An attacker can gain access because of an error in the configuration of a system. For example, the default configuration of some systems includes a "guest" account that is not protected with a password.

Masquerading: In some cases it is possible to fool a system into giving access by misrepresenting oneself. An example is sending a TCP packet that has a forged source address that makes the packet appear to come from a trusted host.

2.2 Intrusion Detection Systems

Intrusion detection systems gather information from a computer or network of computers and attempt to detect intruders or system abuse. Generally, an intrusion detection system will notify a human analyst of a possible intrusion and take no further action, but some newer systems take active steps to stop an intruder at the time of detection [49].

Although there are many possible sources of data an intrusion detection system can use, three types of data were provided to participants in the 1998 Lincoln Laboratory intrusion detection evaluation. Most intrusion detection systems in existence today use one or more of these three types of data. The first of these data sources is traffic sent over the network. All data that is transmitted over an ethernet network is visible to any machine that is present on the local network segment. Because this data is visible to every machine on the network, one machine connected to this ethernet can be used to monitor traffic for all the hosts on the network. During the DARPA evaluation, network traffic was sniffed using a single machine running the tcpdump program [39] to save the network traffic. A second source of data for an intrusion detection system is system-level audit data. Most operating systems offer some level of auditing of operating system events. The amount of data that is collected could be as limited as logging failed attempts to log in, or as verbose as logging every system call. Basic Security Module (BSM) [62] data from a Solaris victim machine was collected and distributed as part of the DARPA evaluation data. A third source of data distributed to the evaluation participants was information about file system state. Daily file system dumps were collected from each of the machines used in the simulation. An intrusion detection system that examines this file system data can alert an administrator whenever a system binary file (such as the ps, login, or ls program) is modified. Normal users have no legitimate reason to alter these files, so a change to a system binary file indicates that the system has been compromised. Although there are many other potential sources of data that can be used by an intrusion detection system to find attacks (such as real-time process lists, logfiles, processor loads, etc.), these three sources (sniffed network traffic, host-level audit files, and file-system state) were provided to participants in the 1998 Lincoln Laboratory DARPA intrusion detection evaluation because they were determined to be the sources most commonly used by the evaluation participants.

After the three types of data were collected and aggregated, the data was distributed to participants via CD-ROM. Once participants obtained this data, each group used its particular intrusion detection system to the find intrusions and abuses that were inserted into the collected traffic. Although the 1998 DARPA evaluation tested only the ability to find attacks offline, some intrusion detection systems can evaluate data in real-time, allowing administrators (or the system itself) to take defensive action against the intruder.

2.3 Strategies for Intrusion Detection

The different approaches that have been pursued to develop intrusion detection systems are described in many papers, including [3][44][63]. Figure 2-1 shows four major approaches to intrusion detection and the different characteristics of these approaches. The lower part of this figure shows approaches that detect only known attacks, while the upper part shows approaches that detect novel attacks. Simpler approaches are shown on the left and approaches that are both computationally more complex and have greater memory requirements are shown towards the right.

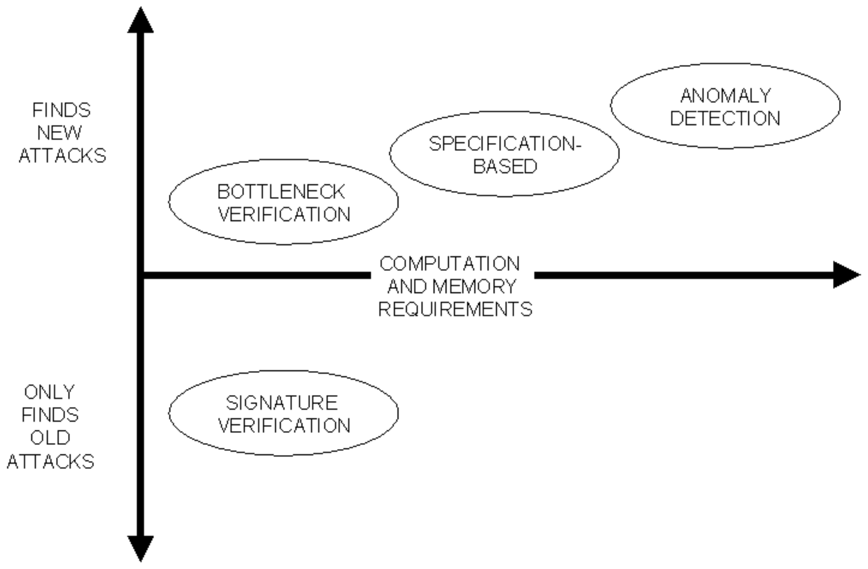


Figure 2-1: Approaches to Intrusion Detection

The most common approach to intrusion detection, denoted as "signature verification" is shown on the bottom of Figure 2-1. Signature verification schemes look for an invariant sequence of events that match a known type of attack. For example, a signature verification system that is looking for a Ping Of Death denial-of-service attack (an oversize ping packet that causes some machines to reboot) would have a simple rule that says "any ping packet of length greater than 64 kilobytes is an attack." Attack signatures can be devised that detect attempts to exploit many possible system vulnerabilities, but a large drawback of this strategy is that it is difficult to establish rules that identify novel types of attacks. The Network Security Monitor (NSM) was an early signature-based intrusion detection system that found attacks by searching for keywords in network traffic captured using a sniffer. Early versions of the NSM [29][28] were the foundation of many government and commercial intrusion detection systems, including

NetRanger [23] and NID [41]. Signature verification systems are popular because one sniffer can monitor traffic to many workstations and the computation required to reconstruct network sessions and search for keywords is not excessive. In practice, these systems can have high false-alarm rates (e.g. 100's of false-alarms per day) because it is often difficult to select keywords by hand that successfully detect real attacks without creating false alarms for normal traffic. In addition, signature verification schemes must be updated frequently to detect new attacks as they are discovered. Recent research on systems which rely on signature verification includes BRO [47] and NSTAT [38].

The approaches shown in the upper half of Figure 2-1 can be used to find novel attacks. This capability is essential to protect critical hosts because new attacks and attack variants are constantly being developed.

Anomaly detection, shown in the upper right of Figure 2-1, is one of the most frequently suggested approaches to detect novel new attacks. Anomaly detection schemes construct statistical models of the typical behavior of a system and issue warnings when they observe actions that deviate significantly from those models. NIDES was one of the first statistical-based anomaly detection systems used to detect unusual user [36] and unusual program [1] behavior. The statistical component of NIDES forms a model of a user, system, or network activity during an initial training phase. After training, anomalies are detected and flagged as attacks. Of course, anomalous behavior does not always signal that an attack is taking place, so anomaly detection systems need to be carefully tuned to avoid high false alarm rates. This level of tuning is only possible if normal user or system activity is stable over time and does not overlap with attacker activity. A user with very regular habits will be easy to model, and any intruder attempting to masquerade as such a user would likely exhibit behavior that deviated significantly from the user's normal activity. The actions of a system administrator, however, might be more irregular and harder to distinguish from the actions of an attacker. In addition, a hacker may be able to slowly change the characteristics that an anomaly detection system considers "normal" by deviating only slightly from normal behavior over a long period of time. After the anomaly detection system had been trained to consider more actions "normal" the attacker could mount an attack and avoid detection. A second disadvantage of anomaly detection schemes is the large computation and memory resources required to maintain the statistical model. Recent research on anomaly detection includes the development of EMERALD [46], which combines statistical anomaly detection from NIDES with signature verification.

Specification-based intrusion detection [39] is a second approach that can be used to detect new attacks. It detects attacks that make improper use of system or application programs. This approach involves first writing security specifications that describe the normal intended behavior of programs. Host-based audit records are then monitored to detect behavior that violates the security specifications. This approach was applied to 15 UNIX system programs and successfully found many attacks [39]. Specification-based intrusion detection has the potential to provide very low false alarm rates and detect a wide range of attacks including many forms of malicious code such as trojan horses, viruses, attacks that take advantage of race conditions, and attacks that take advantage of improperly synchronized distributed programs. Unfortunately, it is difficult to apply because security specifications must be written for all monitored programs. This is difficult because system and application programs are constantly updated. Specification-based intrusion detection is thus best applied to a small number of critical user or system programs that might be considered prime targets for an attack.

The final strategy shown in Figure 2-1 is bottleneck verification. The bottleneck verification approach applies to situations where there are only a few, well defined ways to transition between two groups of states. One example of such a well-defined transition is transitions from a normal user to superuser within a shell. If an individual is in the user state, the only way to legally gain root privileges is by using the su command and entering the root password. Thus, if a bottleneck verification system can detect a shell being launched, determine the permissions of the new shell, and detect the successful use of the su command to gain root access (or, more importantly, the lack of a successful su command), then illegal transitions from normal user to root user can be detected—even if the transition is being made through some novel method that did not exist when the bottleneck verification system was created [65].

Chapter 3

Simulation Network

The goal of the 1998 DARPA Intrusion Detection System Evaluation was to collect and distribute the first standard corpus for evaluation of intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and novel attacks embedded in a large amount of normal background traffic. One roadblock that has discouraged the creation of such a corpus is the reluctance of companies and government agencies to release data collected from operational computer networks. Data collected from an operational computer network is optimal for the evaluation of intrusion detection systems, but this data may contain personal or sensitive information that could not be released to the many parties who conduct intrusion detection research. For this reason, all data in the 1998 DARPA Intrusion Detection System evaluation was synthesized and recorded on a network which *simulated* an operational network connected to the Internet [25][42].

3.1 Modeling an Air Force Local Area Network

The goal of the simulation network (or simnet) was to accurately simulate the network traffic of a Local Area Network that one might find at a United States Air Force facility. Automatically generated traffic used more than 20 network services, including dns, finger, ftp, http, ident, ping, pop, smtp, snmp, telnet, time, and X. In order to accurately model the features of an Air Force network, statistics were measured from months of actual network traffic that was collected from more than fifty Air Force computer networks. Traffic statistics for automatically generated traffic matched average Air Force statistics. The content of email, ftp, web sites, and files was similar to actual documents. Content was generated using open-source documentation or statistical reconstruction from a large set of unclassified documents that preserved both unigram word frequencies and also the frequency of two-word sequences. Generating the data from public sources allowed the data to be distributed without security or privacy concerns.

The 32 different types of attacks that were inserted in the data collected on the simulation network represent attacks that one would expect to see on an Air Force network comprised mostly of UNIX workstations. These attacks were chosen to represent a mixture of different categories of computer attacks, and to provide a mix of older, more recent, and even some novel attacks. In some cases, transcripts of actual Air Force intrusions were used to develop attack scenarios. When such transcripts were not available, attacks were chosen from publicly known attacks and accounts of intrusions on civilian computer systems. Later sections of this thesis describe the attacks in detail.

3.2 Simulation Hardware and Network Topology

The simulation network consisted of two Ethernet network segments connected to each other through a router. A diagram of the network topology is shown in Figure 3-1. The router is located in the top center of this picture and is labeled “CISCO”. Everything to the left of the router in the figure is considered the “outside” of the network and everything to the right of the router is the “inside” of the simulation network. The inside network is connected to one interface of the router and consists of all the computers that

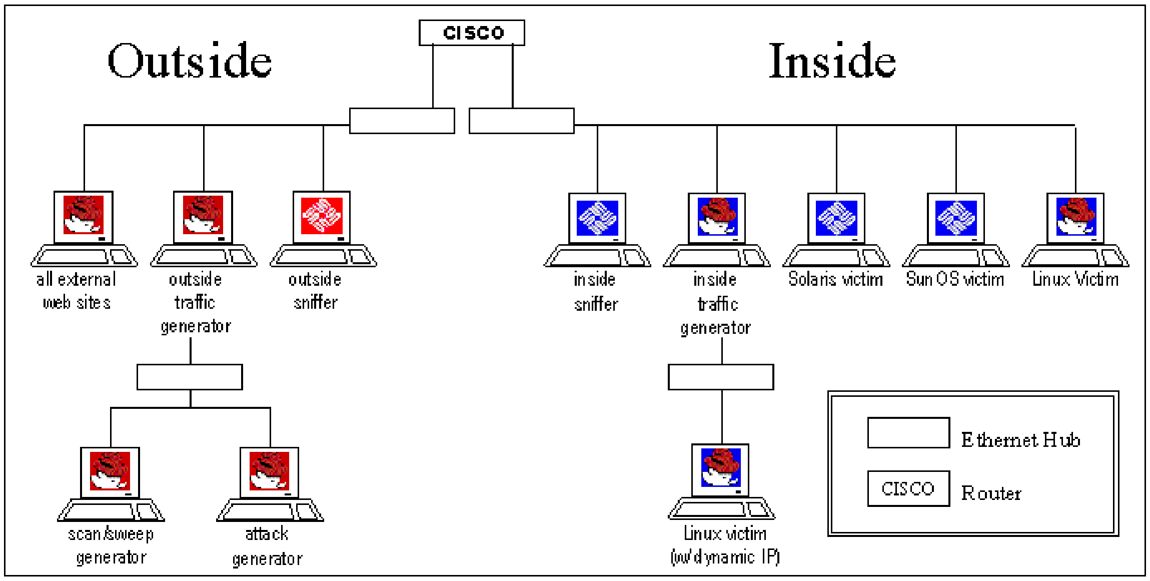


Figure 3-1: Simulation Network Topology

are part of the fabricated “eyrie.af.mil” domain. The computers that model the rest of the Internet are connected to the external interface of the router.

The simulation network included eleven computers. The outside of the network contains a traffic generator (for both background traffic and automated attacks), a web server, a sniffer/recorder, and two machines used for non-automated attack generation. The inside of the network (eyrie.af.mil) consisted of a background traffic generator, a sniffer/recorder, a Solaris 2.5 victim, a SunOS 4.1.4 victim, a Linux 4.2 victim and a Linux 5.0 victim that could dynamically change IP addresses. Although two computers (one inside, one outside) generated all of the background traffic in the simulation, a modification to the operating system of these computers enabled them to act as hundreds of “virtual” machines [26]. The same modification was made to the outside web server so this machine could mimic thousands of web servers. The modification to the operating system that created these virtual machines will be discussed in the next section.

3.3 Simulation Software

The hardware for the simulation network consists of only eleven machines and one router. Custom software allows this simple network to model the interaction of thousands of clients and servers. Without such software the simulation network would consist of hundreds of computers and the evaluation would be impractical in terms of both the cost and time required to maintain the network hardware.

3.3.1 Virtual Machines

The two traffic generators on the simulation network were configured with a modification to the Linux kernel provided by the Air Force Research Laboratory [26]. The IP swapping kernel allows each process that is started on the traffic generator to specify which IP address should be used as the source address for all network traffic generated by that process. At the time of invocation of a new process, an entry is added to a table (called the IP swap table) that maps process id to IP address. Whenever a network packet is generated, the kernel checks the process id of the process that generated the packet and uses the entry in the IP swap table as the source address of the new packet. The modified kernel was an important step in creating the illusion of a virtual machine, but there are several other modifications that support the creation of a single host that can act as many hosts. Daemons that provide network services—including telnetd, ftpd, and login—display banners that identify the machine they are running on. These daemons were modified to give different identification information depending on the destination IP address specified in the request.

3.3.2 Traffic Generation

Automatic traffic regeneration was used to create automated network sessions containing both artificial normal traffic and attacks that appear to be generated by humans working on a system. The traffic regeneration mechanism was designed to be:

- Automatic, requiring no human intervention.
- Reproducible; when repeated, sessions produce identical results.
- Robust, so it can run for long periods without human supervision.

As suggested in [48], the *expect* language was used to automate interaction between a system and a user, allowing autonomous sessions to be run as if a user were typing at a keyboard. Unfortunately, *expect* cannot be used to automate the interaction between a user and a graphical environment (such as X Windows), so any sessions that required interaction with a graphical user interface were run by human actors during the simulation. Because the amount of interaction included in the simulation was so large, creating a separate expect script for each session would have been unwieldy. For this reason, an expect program called the “regenerator” (for “traffic regenerator”) was written that reads information about a particular session from a specially formatted “exs” file. Each “exs” file contains a header and a body. Within the header is information about what time to start a session, the IP address of the local machine, the IP address of the remote machine, and a list of prompts we expect to see during this session. The body of the “exs” file contains one or more prompt-response pairs. For each prompt-response pair the regenerator first waits for the prompt indicated, pauses an indicated amount of time, and then types the command. Each session within the simulation is represented by a single “exs” script. To simulate a day of traffic, a day of “exs” scripts can be collected and passed to the regenerator which then runs all of the sessions. If an error occurs in generating or collecting this traffic, the traffic collection can be repeated by rerunning the regenerator with the same “exs” scripts. The regenerator satisfies the need for automatic, reproducible, and robust traffic generation.

Chapter 4

Exploits

A large sample of actual computer attacks is needed to test an intrusion detection system. These attacks should cover the different classes of attack types and contain exploits for both newly discovered as well as older well-known vulnerabilities. An attack instance might consist of several phases. For example, an attacker might copy a program onto a system, run this program that exploits a system vulnerability to gain root privileges, and then use this root privilege to install a backdoor into the system for later access. This chapter provides some background on the different types of computer exploits, including a taxonomy that can be used to organize these exploits into meaningful groups.

4.1 Sources

Many of the exploits developed for the 1998 DARPA evaluation were drawn from ideas or implementations available from public sources on the Internet. Rootshell [50] is a web-site dedicated to collecting computer exploits and has a sizeable archive of attacks for many popular operating systems. The "Bugtraq" mailing list also frequently hosts "exploit code"—ostensibly released for the purpose of testing one's own system for vulnerability—when a new vulnerability is discussed. A searchable archive of the Bugtraq mailing list can be found on the Internet at http://www.geek-girl.com. Other exploits were created from information released by computer security groups such as

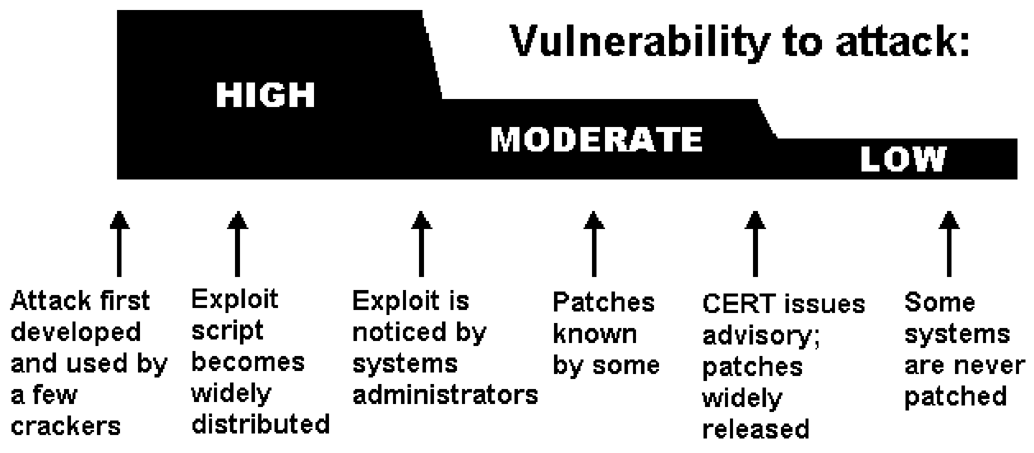

Vulnerability to attack:
HIGH
MODERATE
LOW
Attack first developed and used by a few crackers
Exploit script becomes widely distributed
Exploit is noticed by systems administrators
Patches known by some
CERT issues advisory; patches widely released
Some systems are never patched

Figure 4-1: Vulnerability to a Newly Discovered Attack Decreases with Time

CERT [22] and ISS X-force [35] that frequently release information about new vulnerabilities. Additional sources of information about system vulnerabilities and possible exploits were vendor-initiated bulletins posted by operating system vendors like Sun Microsystems and Redhat Software. These bulletins are released to customers to encourage them to download patches that eliminate a new vulnerability.

New and novel exploits were also created specifically for the purpose of the evaluation. These new exploits are useful for determining how well an intrusion detection system works against novel attacks that were not publicly known at the time the intrusion detection system was developed.

4.2 Age of an Exploit

Each new exploit has a period of time during which it is most dangerous. Figure 4-1 is a simplified illustration of the various phases of a newly discovered exploit. The vertical axis of the figure shows vulnerability to attack and the horizontal axis represents time. As time progresses, more people are made aware of the vulnerability and patch their systems to make them resistant to the exploit. Even after news of a particular vulnerability has become widespread, some systems might not be patched. Some computer systems with less-experienced administrators go years without having widely-known security holes fixed. Some of these older attacks were included in the set of attacks used for the 1998 DARPA evaluation. Intrusion detection systems were generally able to find these older, well-known attacks.

4.3 A Taxonomy for Computer Attacks

A taxonomy for classifying computer attacks was used to choose exploits for the evaluation. A good taxonomy makes it possible to classify attacks into groups that share common properties. Once these groups have been identified, the job of adequately testing an intrusion detection system becomes easier because instead of developing every possible attack we can choose a representative subset from each group. The taxonomy presented here was originally presented in [64]. The features of this taxonomy are:

- Each attack can be reliably placed in one category.
- All possible intrusions have a place in the taxonomy
- The taxonomy can be extended in the future.

This taxonomy was created for the express purpose of testing and evaluating intrusion detection systems. Within the taxonomy, each attack can be categorized as one of the following:

- A user performs some action at one level of privilege
- A user makes an unauthorized transition from a lower privilege level to a higher privilege level.

- A user stays at the same privilege level, but performs some action at a higher level of privilege.

The taxonomy requires a way of describing each privilege level, a way to describe transitions, and a way of categorizing actions. These three requirements are presented in the following sections.

4.3.1 Privilege Levels

The taxonomy defines an approach to ranking levels of privilege. The privilege categories that are applied within this thesis are:

R	Remote network access
L	Local network access
U	User access
S	Root/Super-user access
P	Physical Access to Host

Having privilege at the "Remote network access" level refers to having, via an interconnected network of systems, minimal network access to a target system. "Local network access" represents the ability to read from and write to the local network that the target machine uses. "User access" refers to the ability to run normal user commands on a system. "Root/Superuser access" gives a user total software control of a system. "Physical Access to Host" allows the operator to physically manipulate characteristics of the system (i.e. remove disk drives, insert floppy disks, turn the system off). This list only represents a subset of all possible access levels, but these were the most useful categories for describing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.2 Methods of Transition or Exploitation

An attacker needs to exploit some failure of a security framework in order to perform an attack. The five methods of transition that were explored for the 1998 DARPA evaluation and the single letters (**m**,**a**,**b**,**c**,**s**) used to represent the methods were:

m) Masquerading: In some cases it is possible to fool a system into giving access by misrepresenting oneself. Examples of masquerading include using a stolen username/password or sending a TCP packet with a forged source address.

a) Abuse of Feature: There are legitimate actions that one can perform, or is even expected to perform, that when taken to the extreme can lead to system failure. Example include filling up a disk partition with user files or starting hundreds of telnet connections to a host to fill its process table.

b) Implementation Bug: A bug in a trusted program might allow an attack to proceed. Specific examples include buffer overflows and race conditions.

c) System Misconfiguration: An attacker can exploit errors in security policy configuration that allows the attacker to operate at a higher level of privilege than intended.

s) Social Engineering: An attacker may be able to coerce a human operator of a computer system into giving the attacker access.

An individual attack may use more than one of these methods. For example, a bug in the implementation of the TCP stack on some systems makes it possible to crash the system by sending it a carefully constructed malformed TCP packet. This packet may also have the source address forged so as to avoid identification of the attacker. Such an attack would be exploiting both **m**asquerading and an implementation **b**ug, and it would be possible to detect the intrusion by noting either of these features.

4.3.3 Transitions Between Privilege Levels

To show a transition between two privilege levels the strings for the two levels are written adjacent to one another with the method of transition between them. Two examples are shown in the following table:

Examples:

Attack	*String*	*Description*
Format	U-b-S	User exploits a bug in the format program to become root/superuser
Ftp-write	R-c-U	A user with remote network access exploits a badly configured anonymous ftp server to gain local user access

4.3.4 Actions

There are many actions that can occur as part of a computer attack. Within the taxonomy, actions are represented with a string that represents a category, and a specification string that describes the specific action taken. For example, the string **Probe(Users)**, represents some action taken by an attacker to gather information about the users of a system. The following paragraphs describe the five categories of actions that were used to describe the actions taken during the 1998 DARPA intrusion detection evaluation.

Category	*Specific Type*	*Description*
Probe	Probe(Machines)	Determine types and numbers of machines on a network
	Probe(Services)	Determine the services a particular system supports
	Probe(Users)	Determine the names or other information about users with accounts on a given system
Deny	Deny(Temporary)	Temporary Denial of Service with automatic recovery
	Deny(Administrative)	Denial of Service requiring administrative intervention
	Deny(Permanent)	Permanent alteration of a system such that a particular service is no longer available
Intercept	Intercept(Files)	Intercept files on a system
	Intercept(Network)	Intercept traffic on a network
	Intercept(Keystrokes)	Intercept keystrokes pressed by a user
Alter	Alter(Data)	Alteration of stored data
	Alter(Intrusion-Traces)	Removal of hint of an intrusion, such as entries in log files
Use	Use(Recreational)	Use of the system for enjoyment, such as playing games or bragging on IRC
	Use(Intrusion-Related)	Use of the system as a staging area/entry point for future attacks

Figure 4-2: Summary of Possible Types of Actions.

Probes are actions taken by an attacker to gather information about one or more machine. Probes are represented within the taxonomy by the category label of "**Probe"**. The specific types of probes used in the DARPA evaluation were: (1) Probing a network to see how many and what types of machines are on that network **(Probe(Machines)**), (2) Probing a system to see what services the system supports (**Probe(Services)**), and (3) Probing a system to find out information about user accounts on that system (**Probe(Users)**).

Denial of service attacks are attempts to interrupt or degrade a service that a system provides. These attacks are represented within the taxonomy by the category label "**Deny"**. The classes of denial of service attacks used in the DARPA evaluation were: (1) Temporary denial of service with automatic recovery (**Deny(Temporary)**), (2) Denial of service requiring administrative action for recovery (**Deny(Administrative)**), and (3) Permanent denial of service with total system reconstruction required for recovery (**Deny(Permanent)).**

Another category of attacker actions is the interception of data. Interception of data is represented within the taxonomy by the category label "**Intercept**". The types of data interception actions used in the 1998 DARPA evaluation were: (1) Interception/Reading of files on a file system (**Intercept(Files))**, and (2) Interception of packets on a network (**Intercept(Network)**).

An additional category of action is the alteration or creation of data on a system or network. Actions that involve data alteration or creation are represented with the category label "**Alter"**. The types of data alteration used in the 1998 DARPA evaluation were: (1) Alteration of data stored on a system, such as a password file or any other file (**Alter(Data)**), and (2) Removal of hints of an intrusion, such as entries in log files (**Alter(Intrusion-Traces)**).

The final category of attacker action described in the taxonomy is "use" of a system. Any use of the system that does not fall into the categories described above can be placed in the category represented by the category label "**Use**". The specific ways in which an attacker might use a system that were included in the 1998 DARPA evaluation were: (1) Use of the system by the intruder for enjoyment or recreational purposes such

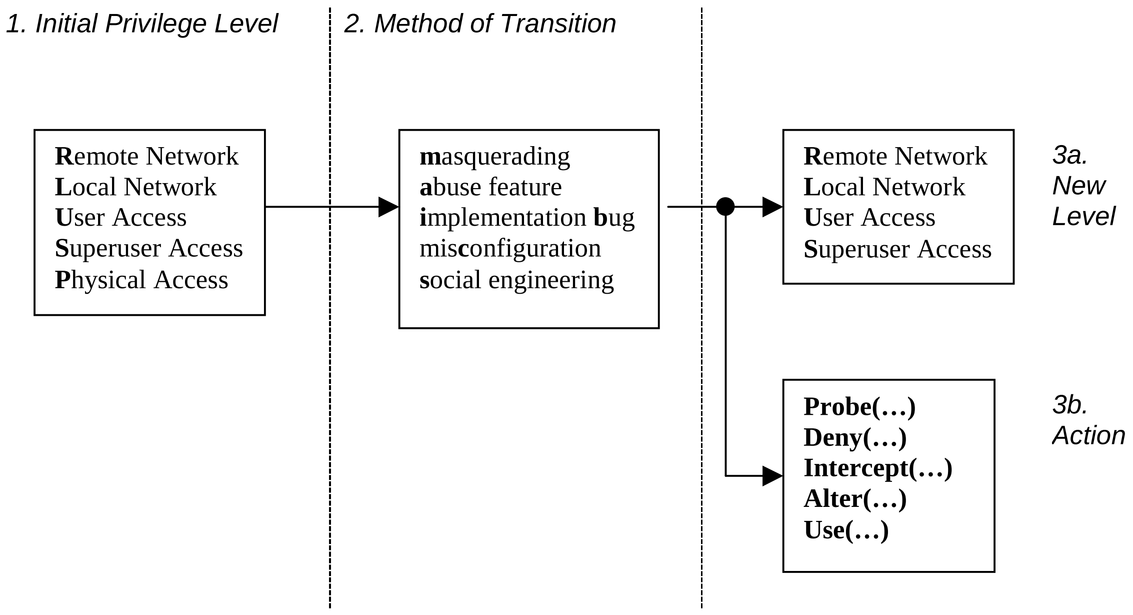


Figure 4-3: A Summary of Possible Attack Descriptions

as playing games or bragging on IRC (**Use(Recreational)**), and (2) Use of a system as a staging ground or entry point for attacks on other systems ((**Use(Intrusion-Related)**).

The action categories and specifications described in the previous paragraphs are summarized in Table 4-2. Each row of this table represents a specific type of action within a category. These specific actions have been grouped according to the categories presented above. The first column of the table is the action category, the second column is the string that represents the action in the taxonomy, and the third column in each row is a description of that particular type of action.

4.3.5 Using the Taxonomy to Describe Attacks

Figure 4-3 gives an overview of the approach the taxonomy uses for the classification of attacks. Each attack classified by this taxonomy is represented as a short alpha-numeric string. The sub-strings in this description are the bold strings from the previous three sections. The initial privilege level is indicated by **R**, **L**, **U**, **S,** or **P** as defined in section 4.3.1, the actions are indicated by any of the strings presented in section 4.3.4, and the method of exploitation is indicated by **m**, **a**, **b, c,** or **s** as defined in section 4.3.2.

In order to describe an attack, select the privilege level that the attacker had before the attack occurred, from the possible choices of **R**emote Network, **L**ocal Network, Local **U**ser, **S**uperuser/Root, and **P**hysical Access. Next, select the method of exploitation, if the method is known. If the method is unknown, then a question mark ("?") is used to indicate the method of exploitation. The possible choices are **m**asquerading, **a**buse of feature, implementation **b**ug, mis**c**onfiguration, or **s**ocial engineering. Finally, either indicate the level of privilege the attacker gained as a result of the exploit (again with **R**, **L**, **U**, **S,** or **P**) or the actions the user performed at the current level of privilege. The taxonomy presented in [64] has additional features that are not discussed here, but the material presented in the section provides a subset of the taxonomy that is useful for the purpose of discussing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.6 Examples

The following table presents three examples that show the correct formatting of the alphanumeric string that specifies an action being performed at a specific privilege level. In a SYN flood (or neptune) attack the attacker sends a stream of SYN packets to a port on a target machine. For a short period of time after these packets have been sent, other users are unable to access the network services provided by that port. In the second example a user runs the crack program to decrypt the password file of a machine that has been compromised. In the third example, the attacker uses the Ffbconfig attack to make a transition from Local **U**ser privilege to Root/**S**uperuser privilege, and then uses this new privilege level to alter the password file on the victim system.

Examples:

Attack	*String*	*Description*
SYN flood	R-a-Deny(temporary)	A user with remote network access temporarily denies service
Cracking passwords	U-Use(Intrusion)	A user with a local account runs a program which attempts to decrypt entries in the password file.
ffbconfig	U-b-S-Alter(Files)	An attacker with a local account uses a bug in the Ffbconfig program to gain root access and alter files.

Chapter 5

Exploits for the 1998 DARPA Evaluation

Figure 5-1 shows the 32 different exploits that were used in the 1998 DARPA intrusion detection evaluation. This table presents the attacks broken up into categories of type and

Figure 5-1: The Attacks Used in the 1998 DARPA Intrusion Detection Evaluation

	Solaris	SunOS	Linux
Denial Of Service (R-Deny)	Apache2	Apache2	Apache2
	Back	Back	back
	Mailbomb	Land	Mailbomb
	Neptune	Mailbomb	Neptune
	Ping Of Death	Neptune	Ping of death
	Process Table	Ping of death	Process Table
	Smurf	Process Table	Smurf
	Syslogd	Smurf	Teardrop
	UDP Storm	UDP Storm	UDP Storm
Remote to User (R-?-U)	dictionary	dictionary	dictionary
	ftp-write	ftp-write	ftp-write
	guest	guest	guest
	phf	phf	imap
	xlock	xlock	named
	xsnoop	xsnoop	phf
			sendmail
			xlock
			xsnoop
User to Super-user (U-?-S)	eject	loadmodule	perl
	ffbconfig	ps	xterm
	fdformat		
	ps		
Surveillance/ Probing (R-Probe)	ip sweep	ip sweep	ip sweep
	mscan	mscan	mscan
	nmap	nmap	nmap
	saint	saint	saint
	satan	satan	satan

vulnerable operating system. The four type categories represent groupings of the possible attack types listed in the taxonomy. These four groups are: Denial of Service (R-?-Deny), Remote to Local User (R-?-U), Local User to Super-user (U-?-S), and Probes (R-?-Probe). The three columns of the table divide the exploits by target platform. Some attacks are listed in more than one column. The Smurf attack, for example, is listed three times—in the Solaris column, the SunOS column, and the Linux column—because all three operating systems are vulnerable to the Smurf attack. The next four chapters present detailed descriptions of each class of attack, and the individual attacks from that class that were included in the 1998 DARPA intrusion detection evaluation

Chapter 6

Denial of Service Attacks

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service (or DoS) attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon. Figure 6-1 provides an overview of the denial of service attacks used in the 1998 DARPA intrusion detection evaluation. Each row represents a single type of attack. The six columns show the attack name, a list of the services that the attack exploits, the platforms that are vulnerable to the attack, the type of mechanism that is exploited by the attack (implementation bug, abuse of feature, masquerading, or misconfiguration), a generalization of the amount of time the attack took to implement, and a summary of the effect of the attack. The following sections describe in detail each of the Denial of Service attacks that were included in the 1998 DARPA intrusion detection evaluation.

Name	**Service**	**Vulnerable Platforms**	**Mechanism**	**Time to Implement**	**Effect**
Apache2	http	Any Apache	Abuse	Short	Crash httpd
Back	http	Any Apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailbomb	smtp	All	Abuse	Short	Annoyance
SYN Flood	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None	Bug	Short	None
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate/ Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/ chargen	All	Abuse	Short	Network Slowdown

Figure 6-1: Summary of Denial of Service Attacks

6.1 Apache2 R-a-Deny(Temporary/Administrative)

Description

The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash [4].

Simulation Details

This exploit was adapted from C code originally posted to the bugtraq mailing list. A C-shell wrapper was also created which executes the apache2 C program in a loop until the server being attacked is no longer responsive.

As soon as the attack was launched the load average (as reported by the “top” program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.

Attack Signature

Every http request submitted as part of this exploit contains many http headers. Although the exact number and value of these headers could be varied by an attacker, the particular version of the exploit which was used in the 1998 DARPA evaluation sent http GET requests with the header “User-Agent: sioux\r\n" repeated 10000 times in each request. The actual content of the header is not important for the exploit—the exploit is only dependent on the fact that http request contains *many* headers. A typical http request contains twenty or fewer headers, so the 10000 headers used by this exploit are quite anomalous.

6.2 Back R-a-Deny(Temporary)

Description

In this denial of service attack against the Apache web server, an attacker submits requests with URL's containing many frontslashes. As the server tries to process these requests it will slow down and becomes unable to process other requests [55].

Simulation Details

The Back attack was implemented as a C shell script that used the Netcat [31] tool to generate network traffic. This shell script was adapted from a script originally posted to the Bugtraq mailing list. Although the number of frontslashes in the URL sent by the shell script could be varied, the number of frontslashes that was determined to be optimal for denial of service against Apache running on Linux 4.2 was between six and seven thousand.

The Back attack causes instances of the httpd process on the victim to consume excessive CPU time. This consumption of the CPU slows down all the system's activities, including responses to network requests. The system recovers automatically when the attack stops.

Attack Signature

An intrusion detection system looking for the Back attack needs to know that requests for documents with more than some number of frontslashes in the URL should be considered an attack. Certainly, a request with 100 frontslashes in the URL would be highly irregular on most systems. This threshold could be varied to find the desired balance between detection rate and false alarm rate.

6.3 Land R-b-Deny(Administrative)

Description

The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [17].

Simulation Details

The land exploit program used in the DARPA evaluation was adapted from a C implementation found at http://www.rootshell.com. The exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address.

Within the simulation, this exploit was run against a Sun SPARC WorkStation running SunOS version 4.1. When a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again.

Attack Signature

The Land attack is recognizable because IP packets with identical source and destination addresses should never exist on a properly working network.

6.4 Mailbomb R-a-Deny(Administrative)

Description

A Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possible causing system failure.

Simulation Details

This exploit was implemented as a perl program that constructed mail messages and connected to the SMTP port of the victim machine directly. The Mailbomb perl program accepted as parameters the e-mail addresses of victims as well as the number of e-mail messages to send.

Although the Mailbomb exploit was used several times throughout the simulation with different parameters, a typical attack would send 10,000 one kilobyte messages (10 megabytes of total data) to a single user. This volume of messages was not enough to adversely effect the performance of the server or cause system failure. As implemented, this attack was more of a nuisance for a particular user than a real threat to the overall security of a server.

Attack Signature

An intrusion detection system that is looking for a mailbomb attack can look for thousands of mail messages coming from or sent to a particular user within a short period of time. This identification is a somewhat subjective process. Each site might have a different definition of how many e-mail messages can be sent by one user or to one user before the messages are considered to be part of a mailbomb.

6.5 SYN Flood (Neptune) R-a-Deny(Temporary)

Description

A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the "tcpd" server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative [13].

Simulation Details

The neptune exploit code used in the simulation was compiled from C code originally posted to the bugtraq archive. The neptune program allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an "infinity" option that would attack all ports).

The neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.

Attack Signature

A neptune attack can be distinguished from normal network traffic by looking for a number of simultaneous SYN packets destined for a particular machine that are coming from an unreachable host. A host-based intrusion detection system can monitor the size of the tcpd connection data structure and alert a user if this data structure nears its size limit.

6.6 Ping Of Death R-b-Deny(Temporary)

Description

The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting [14].

Simulation Details

Several implementations of the Ping of Death exploit can be found at http://www.rootshell.com as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversize ping packets simply by specifying a parameter at the command line (i.e. ping –l 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program.

The Ping of Death attack affected none of the victim systems used in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect.

Attack Signature

An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are longer than 64000 bytes.

6.7 Process Table R-a-Deny(Temporary)

Description

The Process Table attack is a novel denial-of-service attack that was specifically created for this evaluation. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine.

An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed: 1. Open a connection to the target's finger port. 2. Wait for 4 seconds.

3. Repeat steps 1-2. This attack has been attempted against a variety of network services on a variety of operating systems. It is believed that the imap and sendmail servers are vulnerable. Most imap server software contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. With sendmail the situation is reversed. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant [6].

Simulation Details

The version of this exploit used in the simulation was implemented as a perl script that would open connections to a port every **n** seconds, where the port and the number of seconds **n** are specified as run-time parameters. The connections were maintained until a user terminated the script.

The number of connections that must be opened before denial of service is accomplished depends on the size of the process table in the operating system of the victim machine. By using the Process Table attack on the fingerd port as described above, the process table of a Solaris server was exhausted after opening approximately 200 connections (at a rate of one connection every four seconds it took about 14 minutes before the process table was full). After the process table was full, no new process could be launched on the victim machine until the attack was terminated by the attacking program or an administrator manually killed the connections initiated by the attacking script (which is quite difficult to do without launching a new process).

Attack Signature

Because this attack consists of abuse of a perfectly legal action, an intrusion detection system that is trying to detect a process table attack will need to use somewhat subjective criteria for identifying the attack. The only clue that such an attack is occurring is an “unusually” large number of connections active on a particular port. Unfortunately “unusual” is different for every host, but for most machines, hundreds of connections to the finger port would certainly constitute unusual behavior.

6.8 Smurf R-a-Deny(Temporary)

Description

In the "smurf" attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a denial-of-service attack. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim) [18]. The attacker sends ICMP “echo request” packets to the broadcast address (xxx.xxx.xxx.255) of many subnets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP “echo reply” packets to the victim. The smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. In the best case (from an attacker’s point of view), the attacker can flood a victim with a volume of packets 255 times as great in magnitude as the attacker would be able to achieve without such amplification. This amplification effect is illustrated by Figure 6-2. The attacking machine (located on the left of the figure) sends a single spoofed packet to the broadcast address of some network, and every machine that is located on that network responds by sending a packet to the victim machine. Because there can be as many as 255 machines on an ethernet segment, the attacker can use this amplification to generate a flood of ping packets 255 times as great in size (in the best case) as would otherwise be possible. This figure is a simplification of the smurf attack. In an actual attack, the attacker sends a *stream* of icmp “ECHO” requests to the broadcast address of *many* subnets, resulting in a large, continuous stream of “ECHO” replies that flood the victim.

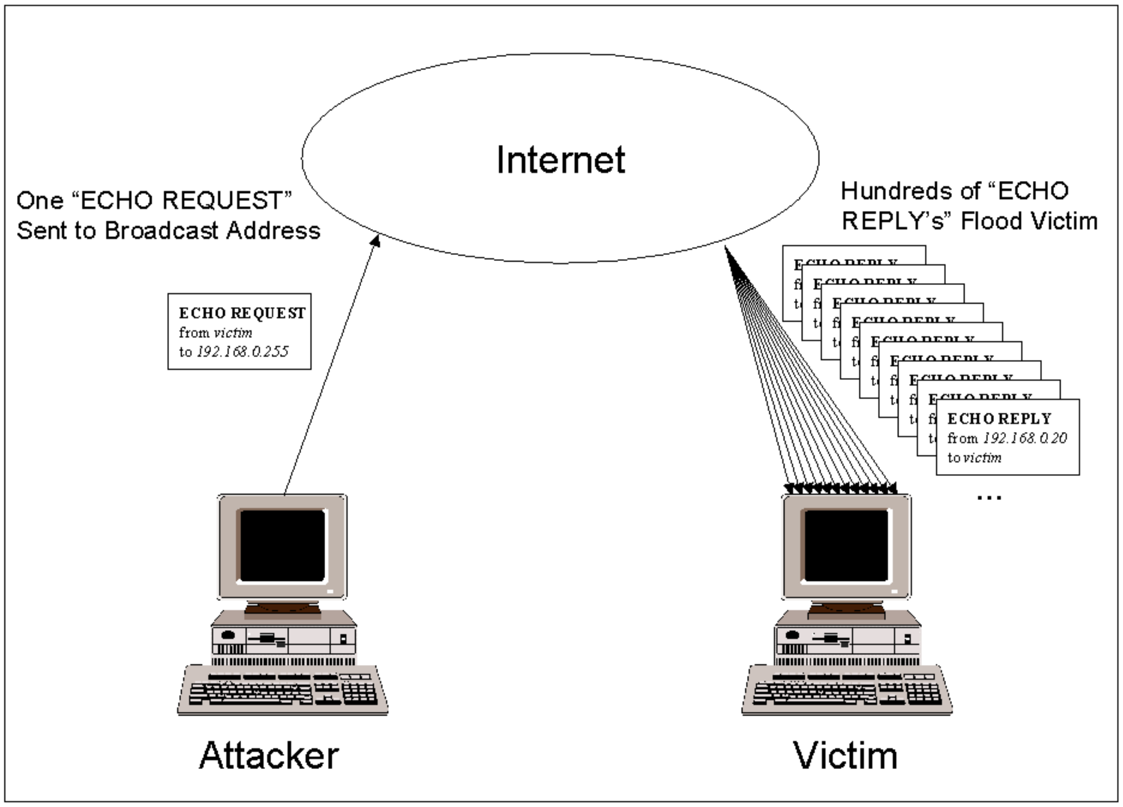

Internet
One "ECHO REQUEST" Sent to Broadcast Address
Hundreds of "ECHO REPLY's" Flood Victim
ECHO REQUEST from victim to 192.168.0.255
ECHO REPLY from 192.168.0.20 to victim
...
Attacker
Victim

Figure 6-2: The Internet is Used to Amplify a Ping Flood and Create a Smurf Attack

Simulation Details

Because the simulation network for the 1998 DARPA evaluation has a flat network topology with only two physical subnets, the smurf attack as described above could not be implemented on the simulation network. For this reason, the "smurfsim" program was developed to recreate the observable effects of a smurf attack. Smurfsim uses the raw socket API to construct ICMP packets with forged source addresses. Smurfsim takes as parameters the IP address of the victim, the number of packets to send, the average percentage of hosts on a subnet that are alive, and a comma-separated list of subnets. The program then randomly constructs a list of hosts that are alive on each of the subnets in the comma-separated list and starts sending "echo reply" packets to the victim, that have been spoofed to look like they originating from the hosts in the list. This behavior is exactly what would occur if an attacker had performed an actual Smurf attack in which "echo request" packets (with the source address spoofed to be that of the victim machine) were sent to the broadcast address of each subnet given in the parameter list.

Several different simulated Smurf attacks were included in the evaluation data. In the most extreme case, the smurfsim program was used to simulate a smurf attack that generated traffic from 100 subnets for a period of one hour. During this period of time the entire simulation network was unresponsive and other network sessions (such as normal users trying to send e-mail, etc) would time out before they could be completed. In all, this particular attack instance generated over two gigabytes of network packets.

Attack Signature

The Smurf attack can be identified by an intrusion detection system that notices that there are a large number of "echo replies" being sent to a particular victim machine from many different places, but no "echo requests" originating from the victim machine.

6.9 Syslogd R-b-Deny(Administrative)

Description

The Syslogd exploit is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault [54].

Simulation Details

The Syslogd exploit used in the 1998 DARPA evaluation was a C program that was originally posted to the Bugtraq mailing list. This code was compiled to create an exploit program that was used to remotely cause the syslogd program to crash on a Solaris 2.5 server. Once syslogd has crashed it must be manually restarted by an administrator for the logging service to be restored.

Attack Signature

The one way to reliably recognize this attack with a network-monitoring intrusion detection system is to notice a packet destined for the syslog port that contains an unreachable source address. Of course, it may not be realistic for an intrusion detection system to check every packet destined for the syslog port to see whether or not the source address is resolvable. If no remote system logging is expected to occur on a particular network, any external syslog messages appearing on this network is likely to be an attack. Finally, a host-based intrusion detection system could be configured to notice the syslog process die because of a segmentation fault.

6.10 Teardrop R-a-Deny(Temporary)

Description

The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments [17].

Simulation Details

The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at http://www.rootshell.com and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.

Attack Signature

An intrusion detection system can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF == 0), with a positive offset greater than N and with a payload of size less than N [5].

6.11 Udpstorm R-a-Deny(Administrative)

Description

A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [10].

Simulation Details

Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo

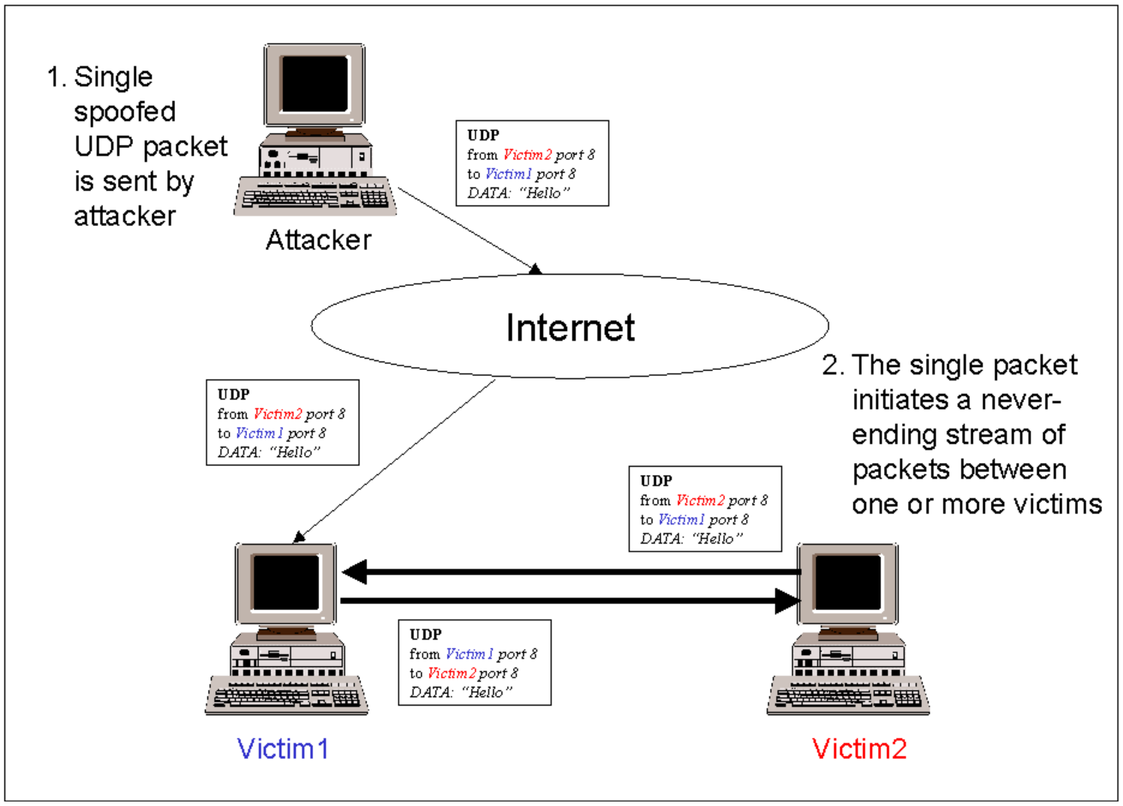

1. Single spoofed UDP packet is sent by attacker
Attacker
UDP from Victim2 port 8 to Victim1 port 8 DATA: "Hello"
Internet
2. The single packet initiates a never-ending stream of packets between one or more victims
UDP from Victim2 port 8 to Victim1 port 8 DATA: "Hello"
UDP from Victim2 port 8 to Victim1 port 8 DATA: "Hello"
UDP from Victim1 port 8 to Victim2 port 8 DATA: "Hello"
Victim1
Victim2

Figure 6-3: UDPStorm is Triggered by a Single Spoofed Packet

port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines.

Attack Signature

This attack can be identified in two ways. First, the single packet that initiates the attack can be recognized because it is a packet originating from outside the network that has been spoofed to appear as if it is coming from a machine inside the network. Second, once the loop of network traffic has been initiated, an intrusion detection system that can see network traffic on the inside of the network can note that traffic is being sent from the chargen or echo port of one machine to the chargen or echo port of another.

Chapter 7

User to Root Attacks

User to Root exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions that was to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Despite the fact that programmers can eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks [2]. Another class of User to Root attack exploits programs that make assumptions about the environment in which they are running. A good example of such an attack is the

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Eject	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Ffbconfig	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Fdformat	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Loadmodule	Any user session	SunOS	Poor Environment Sanitation	Short	Root Shell
Perl	Any user session	Linux	Poor Environment Sanitation	Short	Root Shell
Ps	Any user session	Solaris	Poor Temp File Management	Short	Root Shell
Xterm	Any user session	Linux	Buffer Overflow	Short	Root Shell

Figure 7-1: Summary of User to Root Attacks

loadmodule attack, which is discussed below. Other User to Root attacks take advantage of programs that are not careful about the way they manage temporary files. Finally, some User to Root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously [27]. Although careful programming could eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today.

Figure 7-1 summarizes the User to Root attacks used in the 1998 DARPA evaluation. Note from this table that all of the User to Root attacks can be run from any interactive user session (such as by sitting at the console, or interacting through telnet or rlogin), and that all of the attacks spawn a new shell with root privileges. The following sections describe each of the User to Root attacks that was used in the 1998 DARPA intrusion detection evaluation in greater detail.

7.1 Eject U-b-S

Description

The Eject attack exploits a buffer overflow is the “eject” binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems[60].

Simulation Details

A truncated version of the eject exploit that was used in the 1998 evaluation is shown in Figure 7-2. This exploit was originally posted to the bugtraq mailing list. The exploit script, once compiled, can be run in a command line session on a Solaris server to spawn a shell that ran with root privileges.

Attack Signature

There are several ways that an intrusion detection system might identify this attack. Assuming an attacker already has access to an account on the victim machine and is running the exploit as part of a remote session, a network-based system can look at the contents of the telnet or rlogin session the attacker is using and notice one of several features.

First, assuming that an attacker transmits the C code to the victim machine unencrypted, the intrusion detection system could look for specific features of the source code. For example, an intrusion detection system could look for the string “Jumping to

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <sys/types.h>
4  #include <unistd.h>
5
6  #define BUF_LENGTH 264
7  #define EXTRA 36
8  #define STACK_OFFSET 8
9  #define SPARC_NOP 0xc013a61c
10  u_char sparc_shellcode[] =”

…

17  “;
18
19  u_long get_sp(void)
20  {
21  __asm__("mov %sp,%i0 \n");
22  }
23
24  void main(int argc, char *argv[])
25  {
26  char buf[BUF_LENGTH + EXTRA + 8];
27  long targ_addr;
28  u_long *long_p;
29  u_char *char_p;
30  int i, code_length = strlen(sparc_shellcode),dso=0;
31
32  if(argc > 1) dso=atoi(argv[1]);
33
34  long_p =(u_long *) buf ;
35  targ_addr = get_sp() - STACK_OFFSET - dso;
36
37  for (i = 0; i < (BUF_LENGTH - code_length) / sizeof(u_long); i++)
38  *long_p++ = SPARC_NOP;

…

43  for (i = 0; i < EXTRA / sizeof(u_long); i++)
44  *long_p++ =targ_addr;
45  printf("Jumping to address 0x%lx B[%d] E[%d] SO[%d]\n",
46  targ_addr,BUF_LENGTH,EXTRA,STACK_OFFSET);
47  execl("/bin/eject", "eject", & buf,(char *) 0);
48   perror("execl failed");
49   }
```

Unique Strings that can be Used to Identify an Eject Attack

Figure 7-2: C code for the Eject Exploit

address” on line 45 of the source code or the line “execl (‘/bin/eject’ ; ‘eject’, & buf(char *) 0);” from line 47.

Even if the attacker encrypts the source code, the attack leaves a distinct signature. A segment of the transcript from an actual instantiation of this attack that was used in the simulation is shown below:

```
pascal> /tmp/162562
Jumping to address 0xeffff6a0 B[364] E[400] SO[704]
#
```

An intrusion detection system that saw only these three lines has several clues that an attack has taken place. First, the user's prompt has changed from "pascal>" to "#" without running the su command. Second, the string "Jumping to address" is again printed. Of course, a careful attacker would remove this line from the source code, but simply looking for the string "Jumping to address" would catch the less careful attacker. Finally, a host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument, or by performing bottleneck verification [43] on the transition from normal user to root user and noticing that the user did not make a legal user to root transition.

7.2 Ffbconfig U-b-S

Description

The Ffbconfig attack exploits a buffer overflow is the “ffbconfig” program distributed with Solaris 2.5. The ffbconfig program configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWffbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the ffbconfig program [61].

Simulation Details

This attack is very similar to the eject attack described above. Once again, C code to exploit this vulnerability was posted on the Bugtraq mailing list.

Attack Signature

The means by which an intrusion detection system can identify the ffbconfig attack are similar to those described above for the Eject exploit. An attacker who is exploiting this vulnerability must first transfer the code for the exploit (either C code to be compiled, or pre-compiled code) onto the victim machine, and then run the exploit. As with the eject exploit, there are strings within the source code of the ffbconfig exploit script which identify the attack to a network or host based intrusion detection system. A host-based intrusion detection system can perform bottleneck verification or look for the invocation of the command “/usr/sbin/ffbconfig/” with an oversized argument for the “-dev” parameter.

7.3 Fdformat U-b-S

Description

The Fdformat attack exploits a buffer overflow is the “fdformat” program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program [60].

Simulation Details

Exploit code for this vulnerability was posted to the Rootshell Website [53] in March, 1997. The exploit code was used unmodified for the DARPA evaluation.

Attack Signature

Methods for identifying this attack are nearly identical to those described for the eject and ffbconfig attacks.

7.4 Loadmodule U-b-S

Description

The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine [8].

Simulation Details

The code for the loadmodule exploit script is widely available on the internet. This code is usually in the form of a shell script—it does not need to be compiled before it is run. The steps of the attack are quite simple:

1. Change the value of the internal field separator (or IFS) variable to a slash.
2. Add "." To the front of the PATH variable
3. Copy "/bin/sh" to "./bin"
4. Execute "/usr/openwin/bin/loadmodule a".

When the loadmodule shell script (which is setuid root by default) executes, it attempts to run the command "exec('/bin/a');". Since the IFS variable has been changed to "/" the string "/bin/a" is parsed into two tokens, and the loadmodule script attempts to run the first—"bin". Since the attacker has conveniently put a copy of "/bin/sh" in the current directory and named it "bin", the loadmodule script (which is running as root) will exec "./bin"—giving the attacker a shell with root privileges.

Attack Signature

This attack can be identified either by performing bottleneck verification with a host-based intrusion detection system, or by keyword spotting with a network based intrusion detection system. A simple rule could say that any session which contained the strings “set $IFS=’\/’” and “loadmodule” in close proximity was probably a loadmodule attack. Of course, an attacker could quite easily hide from such a simple rule. Detailed discussion of such methods for hiding is presented in Chapter 11.

7.5 Perl U-b-S

Description

The Perl attack is a User to Root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access [12].

Simulation Details

A perl script that uses this vulnerability to gain root access was made publicly available in August 1996 [51]. The code is only two lines long, and can easily be executed from the command-line. Once this perl script has run, the user will be presented with a new shell that is running with root privileges.

Attack Signature

The methods by which an intrusion detection system could identify a perl exploit attempt are identical to those described above for the loadmodule attack. A host-based intrusion detection system could notice that a root shell was spawned without a legal user to root transition, or a network-based intrusion detection system could look the strings “$>=0; $<=0;” or “exec (‘/bin/sh’);”, which have little valid use except in an exploit attempt.

7.6 Ps U-b-S

Description

The Ps attack takes advantage of a race condition in the version of “ps” distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition [9].

Simulation Details

This exploit is possible because of a combination of the ps program not carefully managing temporary files and a buffer overflow. A shell script that builds a carefully constructed temporary file, creates a C-file, compiles the code and executes the exploit was found at Rootshell.com [52]. Once an attacker has transfers this shell script onto a Solaris victim machine and runs it, a root shell will be spawned for the attacker.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

7.7 Xterm U-b-S

Description

The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the *Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system [21].

Simulation Details

C source code that exploits this vulnerability on Redhat Linux 5.0 systems was found at the Rootshell website [56]. Once again, an attacker can compile this C code, and when the resulting program is run the attacker is given a shell running with root privileges.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

Chapter 8

Remote to User Attacks

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network—but who does not have an account on that machine—exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering—in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse. Figure 8-1 summarizes the characteristics of the Remote to User attacks that were included in the 1998 DARPA intrusion detection evaluation. The following sections provide details of each of these attacks.

8.1 Dictionary R-a-U

Description

The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Dictionary	telnet, rlogin, pop, imap, ftp	All	Abuse of Feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Imap	imap	Linux	Bug	Short	Root Shell
Named	dns	Linux	Bug	Short	Root Shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	smtp	Linux	Bug	Long	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor Keystrokes remotely

Figure 8-1: Summary of Remote to Local Attacks

Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords. Figure 8-2 is a plot of the connections made to the pop3 port of a victim machine during a dictionary attack that is using the pop service to check for valid login/password combinations. The horizontal axis of this plot represents time in minutes, and each line segment in the plot is a single connection to the pop3 service. Lines representing succesive sessions are

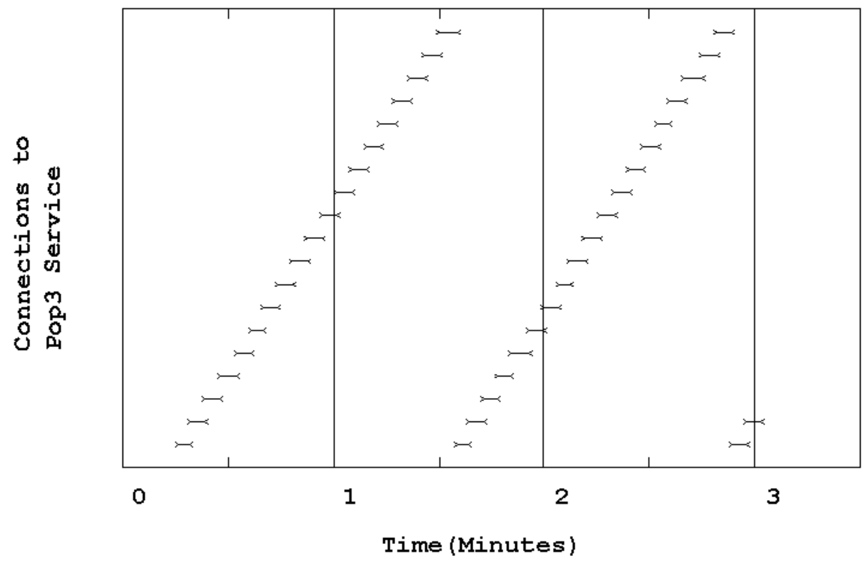


Figure 8-2: Plot of Connections to Pop3 Service During a Dictionary Attack

displaced vertically slightly and wrap around (in this figure at roughly 1.5 minutes). The length of the lines represents the length of the pop sessions. Lines begin with a greater-than sign ">" and end with a less-than sign "<", and thus form an "x" for short sessions. In all, this example dictionary attack consists of 40 attempts to log in, with a 4 second delay between each attempt.

Simulation Details

A perl script that performed automated password guessing on a variety of services was developed specifically for use in our evaluation. The "Netguess" perl script could take in a file of possible username/password combinations, or create guesses for the password based on simple permutations of the username. Within the simulation, this script was used to perform between 10 and 100 login attempts on the telnet, ftp, and pop services.

When the script was successful in gaining access to the system, it would immediately quit and report success.

Attack Signature

An intrusion detection system that finds attempted dictionary attacks needs to know the session protocol of every service that provides username/password authentication. For a given service, the intrusion detection system must be able to recognize and record failed login attempts. Once this functionality is available, detecting dictionary attacks is a matter of setting a detection threshold based on the number of failed login attempts within a given period of time.

8.2 Ftp-write R-c-U

Description

The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system [7].

Simulation Details

This attack was implemented as an expect script which was created explicitly for use in the simulation. This expect script anonymously logged in to the ftp service on the victim machine, created a ".rhosts" file with the string "+ +" in it within the ftp home directory, disconnected from the ftp server, used rlogin to connect back to the server as user "ftp", and finally performed some actions on the victim machine. Creating a ".rhosts" file in the ftp home directory with the entry "+ +" in it allows any user from any machine to rlogin to the victim as user "ftp".

Attack Signature

An intrusion detection system can monitor for this attack by watching all anonymous ftp sessions and assuring that no files are created in the ftp root directory.

8.3 Guest R-c-U

Description

The Guest attack is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [27].

Simulation Details

The Guest attack is a simplified version of the Dictionary attack discussed earlier in this chapter. The same "Netguess" perl script that was used to simulate a Dictionary attack was used to simulate the Guest attack—the only difference between the implementation of the two attacks was the command-line options that were passed to the Netguess program. Whereas the Dictionary attack would try up to a hundred user names and thousands of username/password combinations, the Guest attack would make only a couple of login attempts, using combinations such as "guest/<none>", "guest/guest", "anonymous/<none>" and "anonymous/anonymous".

Attack Signature

Because the Guest attack is essentially a subset of the Dictionary attack, the methods for finding the two attacks are basically the same. An intrusion detection system that is looking for a Dictionary attack already should need only minor tuning in order to find attempts to log in to the guest account.

8.4 Imap R-b-S

Description

The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [16].

Simulation Details

The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit [34]. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.

Attack Signature

The Imap attack can be identified by an intrusion detection system that has been programmed to monitor network traffic for oversized Imap authentication strings.

8.5 Named R-b-S

Description

The Named attack exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the named service can crash the named server or allow an attacker to gain root privileges [19].

Simulation Details

The version of the Named exploit used in the simulation was adapted from a C program originally posted to the Bugtraq mailing list. This program, once compiled, would connect to the named port on a victim machine and overflow a buffer of the named server with instructions that would send an xterm running with root privilege back to the attacker's X console. Because this attack involved interaction with X, all of the Named attacks included in the simulation were run by human actors.

Attack Signature

The Named attack can be identified by watching DNS inverse query requests for messages that are longer than the 4096 byte buffer allocated for these requests within the "named" server.

8.6 Phf R-b-U

Description

The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function escape_shell_cmd() to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [11].

Simulation Details

The Phf attack is quite simple to implement because it requires only the ability to connect to a network socket and issue an http request. Within the simulation, the Netcat [31] program was used to generate this http request. Although this vulnerability allows an attacker to run any command on the server, the command used throughout the simulation was “/bin/cat /etc/passwd”. Using the Phf attack with this command reveals the contents of the victim system’s password file to users with no account on the victim machine.

Attack Signature

To find the Phf attack, an intrusion detection system can monitor http requests watching for invocations of the phf command with arguments that specify commands to be run. Examples of commands that an attacker might attempt to execute by exploiting the phf exploit are: cat /etc/passwd, id, whoami, or xterm.

8.7 Sendmail R-b-S

Description

The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege [15].

Simulation Details

Although this vulnerability was widely known about at the time of the evaluation, no code that exploited this vulnerability had been posted to public forums such as Bugtraq, or Rootshell.com. A significant period of time (one person working for more than two weeks) was spent developing the first known implementation of this exploit explicitly for use in the evaluation. The implementation consists of a carefully constructed mail message which, when sent to the victim machine with a vulnerable version of sendmail, adds a new entry with root privilege to the end of the password file on the victim system. Once this new entry has been added, the attacker can log into the machine as this new user and execute commands as a root user. Figure 8-3 provides an illustration of an instantiation of the Sendmail attack as it was implemented in the simulation. In step 1 of this illustration, the attacker sends a carefully crafted e-mail message to the victim machine. In step 2, the sendmail daemon starts to process this message, overflows one of its buffers, and executes the attacker's inserted commands that create a new entry in the password file. In step 3, the attacker comes back to the victim machine and uses the new password file entry to gain root access to the victim machine and perform some malicious actions.

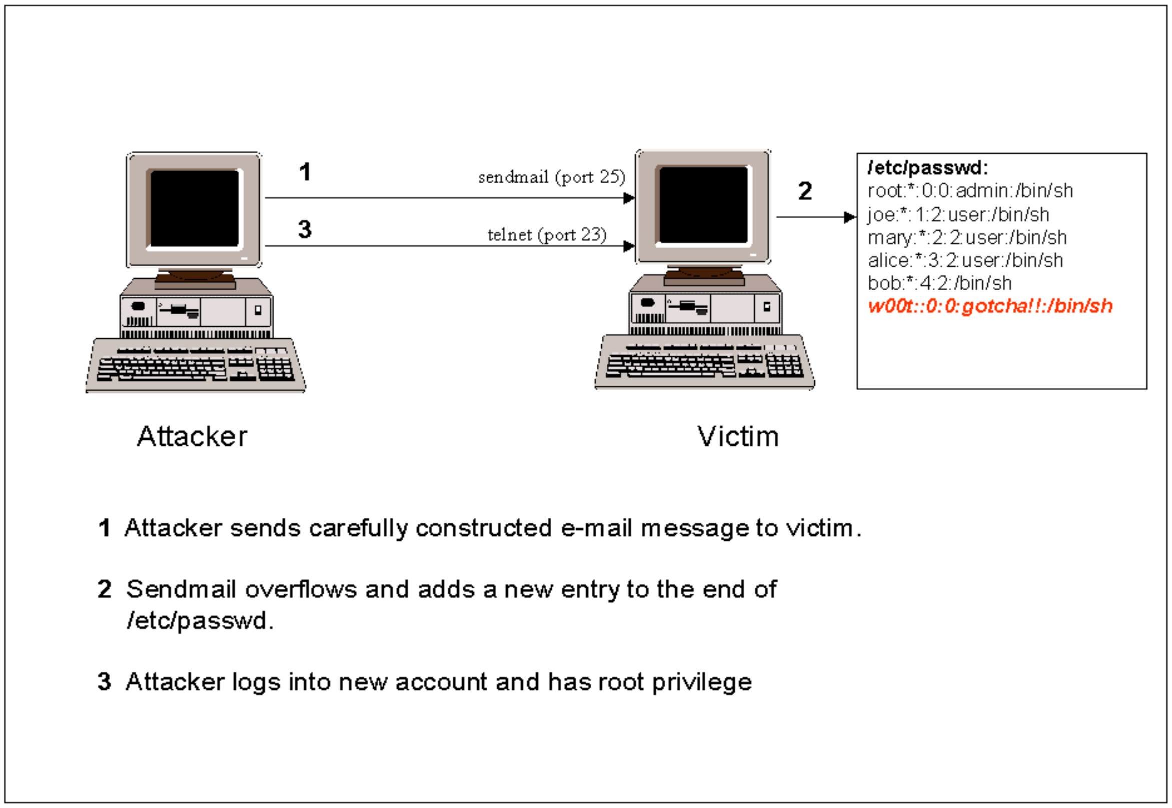

1
sendmail (port 25)
3
telnet (port 23)
2
/etc/passwd:
root:*:0:0:admin:/bin/sh
joe:*:1:2:user:/bin/sh
mary:*:2:2:user:/bin/sh
alice:*:3:2:user:/bin/sh
bob:*:4:2:/bin/sh
w00t::0:0:gotcha!!:/bin/sh
Attacker
Victim
1 Attacker sends carefully constructed e-mail message to victim.
2 Sendmail overflows and adds a new entry to the end of /etc/passwd.
3 Attacker logs into new account and has root privilege

Figure 8-3: Illustration of the Sendmail Attack.

Attack Signature

The Sendmail attack overflows a buffer in the MIME decoding routine of the sendmail program. In order for an intrusion detection system to identify a Sendmail attack it must monitor all incoming mail traffic and check for messages that contain a MIME header line that is inappropriately large.

8.8 Xlock R-cs-Intecept(Keystrokes)

Description

In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing “xhost +”), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.

Simulation Details

This attack was created specifically for use in the evaluation. A special version of the xlock program was created by making small modifications to the publicly available source code for the xlock program. The standard version of xlock will not display on a remote display, and does not save or output the password that the user enters. A modified version of xlock was created that allows the attacker to display the screen saver on a remote display and returns the password entered by the victim. Because of the required interaction with the X server, the Xlock attack was always run by a human actor.

Attack Signature

Two factors make this attack quite difficult for an intrusion detection system to identify. First, this attack is a spoofing attack that does not abuse a bug in the system, but relies on assumptions by the person who is currently using the system. Second, this attack is embedded in the X protocol, and an intrusion detection system that is trying to identify this attack must understand and parse these X communications in order to identify the

Xlock attack. One non-optimal way of dealing with these difficulties, is to program an intrusion detection system to identify as suspicious any X traffic that is originating from an unknown machine that is destined for a machine being monitored.

8.9 Xsnoop R-c-Intercept(Keystrokes)

Description

In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.

Simulation Details

The Xsnoop program used in the simulation was adapted from C code originally posted to the Bugtraq mailing list. The Xsnoop program runs locally on the attacker's machine. The program connects to the victim's X Server and requests to be notified of all X KeyPress Events. If the attacker has permission to make this request (as would happen if the victim had typed "xhost +", or if the victim's X Server is configured to allow connections from all hosts by default) the Xsnoop program will be sent all KeyPress events that occur on the victim X Server. The Xsnoop program then uses the KeyPress events to provide the attacker with a view of all the keystrokes the victim. Because this attack required interaction with the X Server, it was always run by a human actor.

Attack Signature

An network based intrusion detection system can identify the Xsnoop attack by parsing the X protocol information in packets destined for remote X clients and noting that X KeyPress events are being transmitted to a remote machine. Because the Xsnoop attack results from bad security policy (leaving an X Server unsecured) and not simply a bug, the presence of these packets alone does not signify that an attack is taking place. The intrusion detection system must know whether the security policy of the machine being monitored allows unauthenticated X connections from anywhere.

Chapter 9

Probes

In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities [27]. These network probes are quite useful to an attacker who is staging a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools (satan, saint, mscan) enable even a very unskilled attacker to very quickly check hundreds or thousands of machines on a network for known vulnerabilities. Figure 9-1 provides a summary of the probes that are discussed in the remainder of this chapter. The

Name	**Service**	**Vulnerable Platforms**	**Mechanism**	**Time to Implement**	**Effect**
Ipsweep	ICMP	All	Abuse of Feature	Short	Finds active machines
Mscan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Nmap	many	All	Abuse of Feature	Short	Finds active ports on a machine
Saint	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Satan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities

Figure 9-1: Summary of Probes

following sections describe in detail each of the probes that was used in the 1998 DARPA intrusion detection evaluation.

9.1 Ipsweep R-a-Probe(Machines)

Background

An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.

Simulation Details

There are many methods an attacker can use to perform an Ipsweep attack. The most common method—and the method used within the simulation—is to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy—the sweeps were performed linearly, quickly and from a single source.

Attack Signature

An intrusion detection system looking for the simple Ipsweep used in the simulation can look for many Ping packets, destined for every possible machine on a network, all coming from the same source.

9.2 Mscan R-a-Probe(Known Vulnerabilities)

Description

Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities [20].

Simulation Details

The Mscan program used in the simulation was compiled from source code found at [57]. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers.

Attack Signature

The signature of this attack will vary depending on which vulnerabilities are being scanned for and how many machines are being scanned. In general, an intrusion detection system can find an Mscan attack by looking for connections from a single outside machine to the ports listed above on one or more machines within a short period of time.

9.3 Nmap R-a-Probe(Services)

Description

Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscans—options include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.

Simulation Details

At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand.

Attack Signature

The signature of a portscan using the Nmap tool varies widely depending on the mode of operation selected. Despite this variance of modes, all portscans share some common features. A portscan can be recognized by noting that network packets (whether via TCP or UDP, or via only FIN packets or only SYN packets) have been sent to several (or more) ports on a victim or group of victims within some window of time. Two factors complicate the identification of portscans. First, a portscan can happen very slowly. An attacker who is patient could probe one port per day. Current intrusion detection systems do not keep enough state to recognize a portscan happening over such a long period of time. With the amount of network traffic sent over a typical network, keeping enough active state within the intrusion detection system to recognize one connection per day for one hundred days as a one hundred day long portscan is simply not practical. Second, the connections don't necessarily all have to come from the same host. A group can perform a coordinated scan with each member scanning only a subset of machines or ports. By combining these methods, a group could perform a "low/slow" portscan that would be very hard to recognize.

9.4 Saint R-a-Probe(Known Vulnerabilities)

Description

SAINT is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services. The information gathered includes the presence of various network information services as well as potential security flaws. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. Although SAINT is not intended for use as an attack tool, it does provide security information that is quite useful to an attacker [58].

Simulation Details

SAINT is distributed as a collection of perl and C programs and is known to run on Solaris, Linux, and Irix systems. Within the simulation, the Saint program was run from a Linux traffic generator and was used to probe several victim machines for vulnerabilities. SAINT's behavior is controlled by a configuration file which allows the user to specify several parameters. The most important parameters are the list of machines to scan, and how heavily to scan these machine (light, normal, or heavy). In light mode, SAINT will probe the victim for dns and rpc vulnerabilities and will look for unsecured NFS mount points. In normal mode SAINT will also check for vulnerabilities in fingerd, rusersd, and bootd, and will perform a portscan on several tcp (70, 80,ftp, telnet, smtp, nntp, uucp) and udp (dns, 177) ports. Heavy mode is the same as normal mode except that many more ports are scanned.

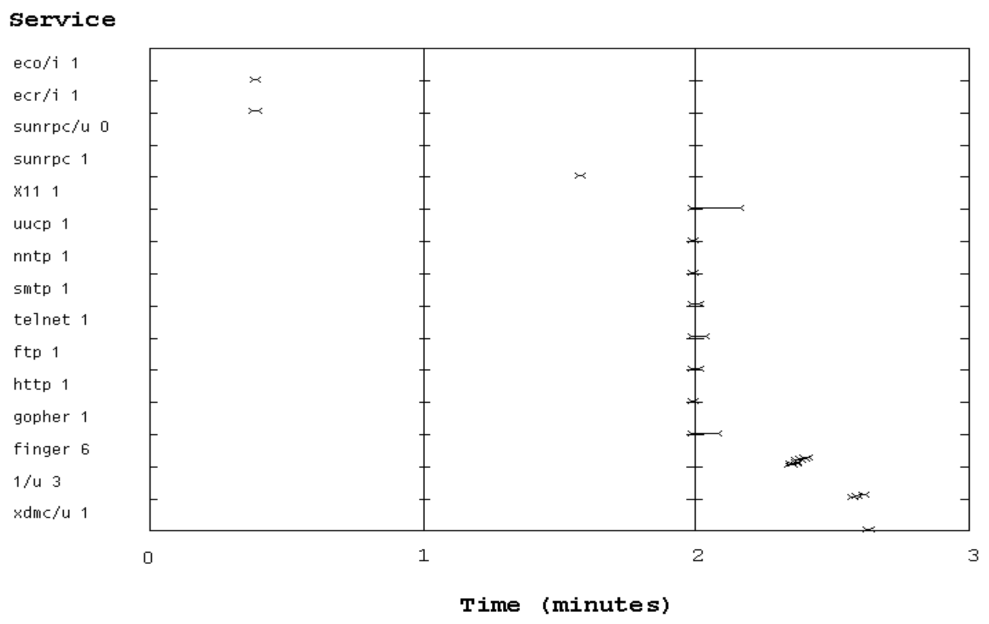


Figure 9-2: Plot of Connections During a Medium Level Saint Scan

Attack Signature

A Saint scan of a network leaves a distinct signature that will vary depending on the level of scanning being performed. The Saint program performs each scan in a nearly deterministic fashion. To identify a Saint scan, an intrusion detection system needs to be able to recognize the distinct set of network traffic the scan creates. Figure 9-2 is a plot that provides a graphical view of this signature. The horizontal axis of this plot represents time in minutes, and the different services probed are presented along the vertical axis. The names of the services are shown on the left side of this plot, and connections for each service are plotted within each named region. The numbers after the service names are the number of separate tcp connections or of udp or icmp packets. Names ending in "/i" indicate that packets use the icmp protocol and names ending in "/u" indicate that packets use the udp protocol. All other services use the tcp protocol.

Each line segment represents a connection to a service. This plot shows the unique signature of a medium level Saint scan. Because this signature does not change significantly across multiple instantiations of the Saint attack, an intrusion detection system that has been trained to recognize the pattern of connections shown in Figure 9-2 will probably detect other Saint attacks.

9.5 Satan R-a-Probe(Known Vulnerabilities)

Description

SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tools checks for are slightly different [24].

Simulation Details

Like SAINT, SATAN is distributed as a collection of perl and C programs that can be run either from within a web browser or from the UNIX command prompt. SATAN supports three levels of scanning: light, normal, and heavy. The vulnerabilities that SATAN checks for in heavy mode are:

- NFS export to unprivileged programs
- NFS export via portmapper
- NIS password file access
- REXD access
- tftp file access
- remote shell access
- unrestricted NFS export
- unrestricted X Server access
- write-able ftp home directory
- several Sendmail vulnerabilities
- several ftp vulnerabilities

Scans in light and normal mode simply check for smaller subsets of these vulnerabilities.

Attack Signature

A SATAN scan of a network can be recognized by the consistent pattern of network traffic the program creates. The checks for the vulnerabilities listed above are always performed in the same order.

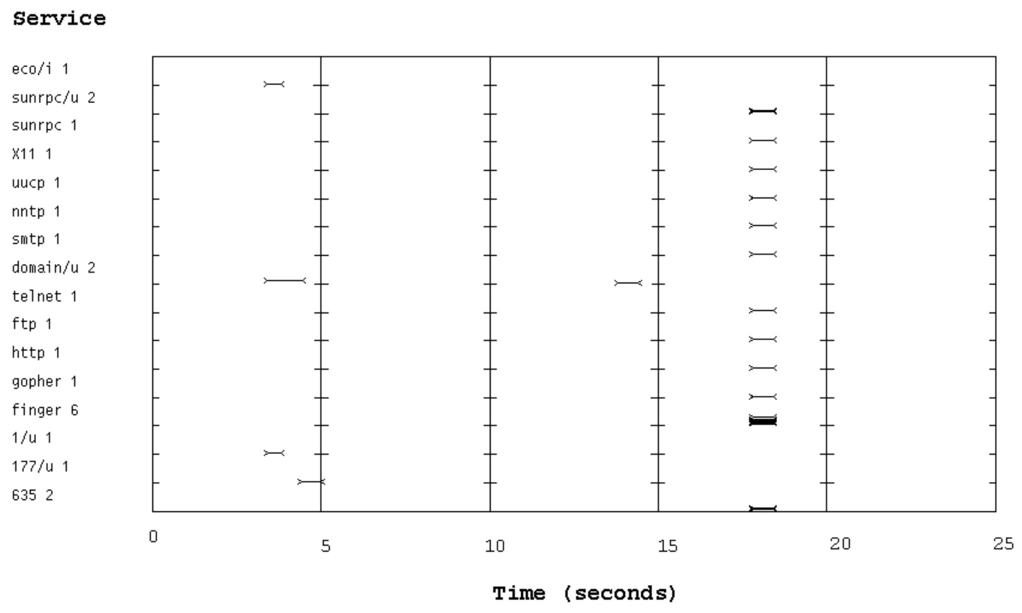


Figure 9-3: Plot of Connections During a Medium Level Satan Scan

Figure 9-3 shows a plot of the services probed during an example medium level Satan scan. The horizontal axis of this plot represents time in seconds and the various services that are probed are presented on the vertical axis. Each line segment in the plot represents a single connection to a service. Every medium level Satan scan will have a signature very similar to that shown in Figure 9-3.

Chapter 10

Realistic Intrusion Scenarios

An attack against a computer system can consist of several phases. An attacker might run several probes against a system looking for weaknesses, then use information gained from these probes to exploit some security vulnerability and gain access to the system. Later an attacker can use this access to install a backdoor in the system, or plant a "logic bomb" in some program that causes damage at some point in the future. The actions of a single intruder might be spread out over the course of weeks or even months. Some attackers make no attempt to hide their intrusions from systems administrators while others are very careful to cover their tracks. In order to accurately test an intrusion detection system's effectiveness in finding real attackers, it was necessary to not only develop a database of realistic exploits and probes, but to combine these exploits and probes with probable attacker actions and realistic goals to generate complete attack scenarios. This chapter describes how these various parameters of attacker behavior were varied to create a realistic simulation of the wide variety of attacker behavior that is manifested by actual intrusions.

10.1 Attack Scenarios

Most of the attacks that were included in the evaluation consist of a single session, or a few sessions that all occur within some short period of time. In the real world, an attacker often has a goal in mind, and sometimes this goal cannot be achieved in a single session. Several complex scenarios were added to the collected data in an attempt to create a better simulation of real attacker behavior. Each scenario consists of a planned sequence of sessions (sometimes over the course of a week or more) that represent the actions of a single individual in pursuit of a goal. The following subsections describe each scenario in detail.

10.1.1 Cracker

The default scenario that occurred in 95% of sessions is that a curious cracker was trying to gain access to a machine just to prove that it could be done. Usually these crackers are simply trying to break into as many machines as they can, and may install a backdoor or download the password file in order to guarantee that they can access the machine again. Crackers are represented as individuals of different skill levels, some perform all of their actions in the clear, while others are aware that an intrusion detection system is present and take actions to avoid detection.

10.1.2 Spy

The spy is an information collector who comes back to a compromised machine several times to collect information. A spy might be looking for confidential data files or reading user's personal mail. A spy will take steps to minimize the possibility of detection.

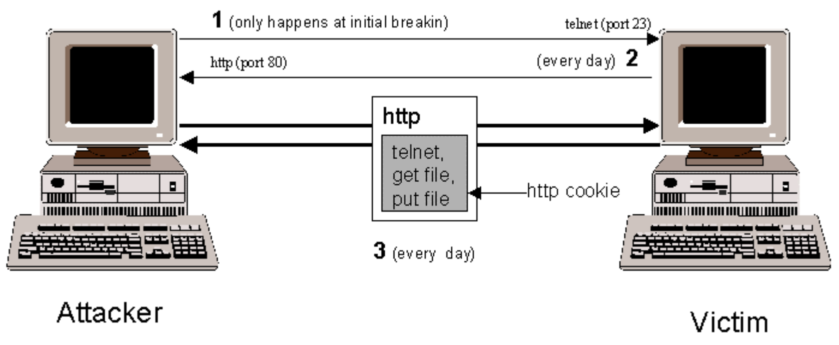
10.1.3 Rootkit

The rootkit scenario could be viewed as an extension of the default Cracker scenario.

A rootkit is a collection of programs that are intended to help a hacker maintain access to a machine once it has been compromised. A typical rootkit consists of a sniffer, versions of login, su, and other programs with backdoors which allow for access, and new versions of ps, netstat, and ls that hide the fact that a sniffer is running and hide files in certain directories. Once the rootkit has been installed, the attacker comes back several times to download the sniffer logs.

10.1.4 Http Tunnel

The Http Tunnel scenario was originally developed as a method of defeating a firewall and for continuing to access a system while minimizing the chance of detection. The http tunnelling tools used in this scenario were developed specifically for use in this evaluation. Figure 10-1 provides an overview of this scenario. Assuming that a hacker is able to penetrate a firewall once (perhaps by sending an email with the executable content of the client in it or by dialing into a modem) a server program can be installed that masquerades as a normal user browsing web pages. Once this server has been installed, the hacker can issue requests to execute commands or transfer files with interaction happening in web traffic between the server and the hacker. For the simulation, the data was exchanged through cookies that rode along with a web request. This general method is very flexible however, and the data could have been tunneled using any cryptographic or steganographic technique. Tunneling through http was chosen with hopes that the large amount of http traffic most networks see in a typical day would obscure the actions of the attacker.



1 Attacker telnet's into victim and installs the httptunnel client, and configures the client to start up once a day. Attacker also sets up the httptunnel server on his own machine which will listen for requests from the client.

2 Client (on the victim) initiates request to server once a day.

3 Server and client continue to communicate--tunneling things like telnet, file transfers through the http protocol.

Figure 10-1: Attacker Uses Http to Tunnel Through a Firewall

10.1.5 SNMP Monitoring

An attacker who has guessed the SNMP community password of a router will then be able to monitor the traffic levels on that router, and may be able to issue commands to the router to change default routes or allow connections from a previously forbidden host or network.

10.1.6 Multihop

Some intrusion detection systems monitor traffic immediately outside of a router and only see traffic going into or coming out of that network. The multihop scenario was designed to test whether these systems could find attacks where an attacker first breaks into one inside machine, and then uses this inside machine for further attacks on the rest of the machines on the network. This would be a very effective means of launching a Denial of Service attack undetected, as an intrusion detection system which sits just outside the network being monitored would not see a Denial of Service attack that originated from the internal network.

10.1.7 Disgruntled/Malicious User

These sessions simulate an attacker who is not interested in collecting information from a system or gaining access to a system, but is simply interested in doing damage. This is a common threat in operational computer networks [32]. Within the simulation, one malicious user re-formatted the primary disk partition of a victim machine.

Chapter 11

Stealthiness and Actions

In addition to varying the methods and intentions of the simulated attackers, attention was given to the extent to which attackers tried to hide their actions from either an individual who is monitoring the system, or an intrusion detection system. There are several ways that attackers can reduce their chances of being detected by the administrator of a network. Skilled attackers might try to cover their tracks by editing system logs or resetting the modification date on files that they replaced or modified. These actions are generally intended to reduce the chance of detection by a human administrator. Attackers may also be aware that an intrusion detection system is monitoring a network, and may try to hide from the intrusion detection system as well. Methods for being stealthy vary depending on the type of attack.

11.1 Avoiding Detection of Denial of Service (R-Deny)

Denial of Service attacks are difficult to make stealthy. One method an attacker can use to hide a denial of service attack is to gain the cooperation of a large group and break up the attack so pieces of it are coming from several different sources. Another method an attacker can use is to send thousands of packets with different spoofed source-addresses. Sending these spoofed packets will not make identifying the attack any harder, but they will make it more difficult to track down and stop the attacker because the victim has no way of knowing which of the thousands of addresses the actual attack is coming from. Both of these methods do not make the attack any harder to detect, but simply reduce the chances that the attacker will be caught. No stealthy denial of service attacks were included in the 1998 DARPA intrusion detection evaluation.

11.2 Avoiding Detection of Probes (R-Probe)

Several methods can be used either to hide the fact that a probe is occurring, or obscure the identity of the party who is performing the probe. The following paragraphs describe methods of increasing stealth that were used in the probes included in the 1998 DARPA intrusion detection evaluation.

Scan Slowly and Randomly: If an attacker wants to hide the fact a probe is occurring, the probe can be configured to occur slowly and probe ports or machines in a non-linear order. An intrusion detection system will have a very hard time identifying one stray connection per hour to a random port as a port sweep initiated by an attacker.

Probe With Half-Open or Other Unlogged Connections: Another method of scanning stealthily is to probe with half-open connections. A connection for which the three-way TCP handshake is never completed will not be logged by the operating system. There are several tools available that will perform this type of half-open (FIN) scanning of a network.

Use an Intermediate Machine to Obscure the Real Source of the Scan: One way attackers can hide their identity is to use an ftp bounce probe. Some ftp servers will allow anyone to tell them to send data to a particular port on a particular machine. An attacker can look at the response the ftp server gives from such a request and ascertain whether that port is listening on the victim machine. The portscan will appear to be coming from an anonymous ftp server, and this simple step may be enough to assure that the party who is really doing the scanning is never identified.

11.3 Avoiding Detection of User to Root (L-?-S) Attacks

There are many ways that User to Root attacks can be made stealthy. The following paragraphs discuss each of the methods that were used to make a number of the User to Root attacks in the 1998 DARPA intrusion detection evaluation stealthy.

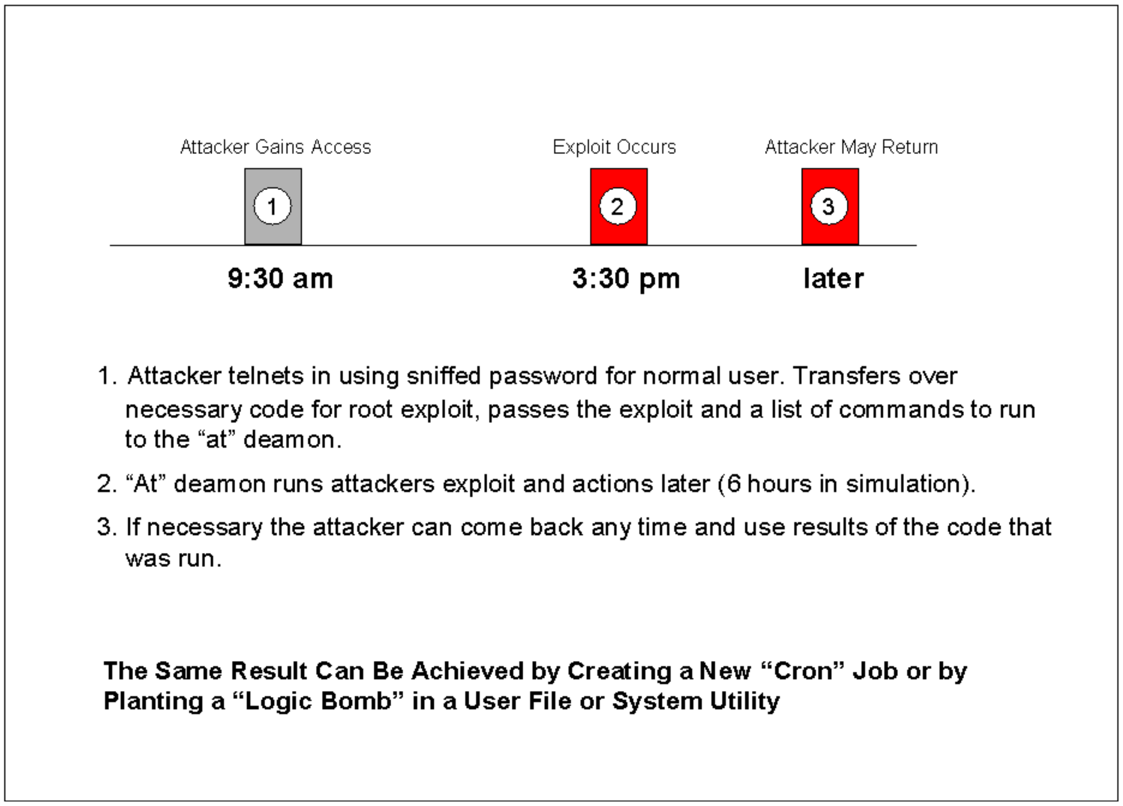
Keyword Hiding: Some intrusion detection systems that attempt to detect illegal User to Root transitions rely on keyword spotting to detect intruders. For example, if the system observes the text of the C source code for the publicly available Eject exploit in a telnet or rlogin session, it will flag this session as being suspicious. By uuencoding or Mime encoding the text of the code before sending it over the network connection, an attacker would avoid detection by such a system.

Output Hiding: It is possible to identify attacks by looking that the output that is displayed on the terminal when the exploit is run. An attacker can avoid detection via this mechanism by sending all the output of commands that are run to a file and encoding (again with some method like ROT13, uuencode, or gzip) the file before displaying or transferring it.

Command Hiding: A system might also look for an attacker to run some command that only the superuser should be able to run, such as displaying the contents of the shadow password file. The invocation of a command that the attacker wishes to hide from someone who is looking for certain suspicious commands or actions can be obfuscated by using glob constructs and character replacement. Instead of typing the command "cat /etc/passwd", the attacker can issue the command "/[r,s,t,b]?[l,w,n,m]/[c,d]?t

/?t[c,d,e]/*a?s*”. When the shell tries to interpret this input string it will do replacement of the glob characters and find that the only valid match for this string is “/bin/cat /etc/passwd”.

Delayed Attack: An attacker can also avoid detection (or at least reduce the chances of identification), by separating the time of exploit from the initial time of access. This can be accomplished by submitting a job to the “at” or “cron” daemon which will run the attack at some later time. An attacker can log in as a normal user (which would not be noticed by an intrusion detection system) and submit a shell script to the “at” daemon which would execute any actions that the attacker desired three weeks (or six months, or five years!) after the attacker initially logged into the system. Figure 11-1 provides a graphical illustration of a delayed attack scenario. This scenario is similar to one that was used in the evaluation. First, at time segment 1 in the figure, the attacker uses a sniffed password to log into the victim machine as a normal user. This action will not be noticed by an intrusion detection system. Before logging out, the attacker submits a job to the cron or at daemon that performs a User to Root exploit and some malicious actions at some point in the future. At time segment 2 in the figure, the submitted script is run and the exploit occurs. Later, at time segment 3 the attacker can come back and log into the machine as a normal user to collect the output from the commands run during the second segment, again without being noticed. The only step in this process that is easily detectable by an intrusion detection system is step 2 when the actual exploit occurs. However, during this time segment the attacker is not logged in to the system. In order to successfully identify this attack, an intrusion



1. Attacker telnets in using sniffed password for normal user. Transfers over necessary code for root exploit, passes the exploit and a list of commands to run to the "at" deamon.
2. "At" deamon runs attackers exploit and actions later (6 hours in simulation).
3. If necessary the attacker can come back any time and use results of the code that was run.

The Same Result Can Be Achieved by Creating a New "Cron" Job or by Planting a "Logic Bomb" in a User File or System Utility

Figure 11-1: Attacker Can Use "at" to Temporally Disassociate the Time of Access and Time of Attack

detection system would need to correlate the earlier session to the malicious actions that occur when the attacker is not logged in.

Figure 11-2 is a transcript from an actual stealthy eject attack used in the simulation. This attack session contains the keyword hiding, output hiding, and command hiding techniques discussed above. Lines 10 through 16 contain the output of the attacker transferring the exploit files to the victim system. The files have been uuencoded to help avoid detection of the attack by a system that is looking for keywords that might be located in the text of the exploit files.

```
1  UNIX(r) System V Release 4.0 (pascal)
2
3  login: alie
4  Password:
5  Last login: Thu Jul 30 09:15:04 from 135.8.60.182
6  Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
7 pascal>
8 pascal> which gcc
9 /bin/gcc
```

```
10 pascal> uudecode
11 begin 644 /tmp/16256.c
12 M("‘@("\J"B‘@("!3;VQA<FES(#(N-2XQ("T@=&AI<R!E>’!L;VET960@=V%S
13 M(&-O;7!I;&5D(&]N(%-O;&%R:7,R+C0@86YD(’1E<W1E9"!O;@H@("‘@,BXU
14 ‘
15 end
16  <FILE TRANSFER TRUCATED FOR EXAMPLE>
```

Keyword Hiding

```
17 pascal>
18
19 pascal> /bin/gcc -o /tmp/162562 /tmp/16256.c
20 pascal> which gcc
21 /bin/gcc
22 pascal> /bin/gcc -o /tmp/162563 /tmp/16256.c
23 pascal> uudecode
24 begin 644 /tmp/16256.c
25 M(VEN8VQU9&4@/’5N:7-T9"YH/@IV;VED"FUA:6XH:6YT(&%R9V,L(&-H87(@
26 M*F%R9W9;72D*>PH@(’-E=’)E=6ED*#‘L,"D["B‘@97AE8VPH(B]B:6XO=&-S
27 0:"(L(G1C<V@B+#‘I.PI]"@‘‘
28 ‘
29 end
30 pascal>
31
32 pascal> /tmp/162562
33 Jumping to address 0xeffff6a0 B[364] E[400] SO[704]
34 #
```

```
35 #ls | /u?r/l?c?l/b?n/g?ip | /b?n/u?e?c?d? gfd
36 begin 644 gfd
37 B’XL( %@:P34  RM)+2XQY"H!D7K)8-J("P"DGT’[%      !
38
39 end
40 # /u?r/b?n/i? | /u?r/l?c?l/b?n/g?ip | /b?n/u?e?c?d? asd
41 begin 644 asd
42 M’XL( %L:P34  RO-3+$U,C0TT$C,R4S55$@’<@T-##1*BU.+BC454DN!? .-
43 /HOS\$DTN ’CL’1@K
44
45 end
46 # /bi?/c?t /et?/p?s?w? | /u?r/l?c?l/b?n/g?ip | /b?n/u?e?c?d? hghhgf
47 begin 644 hghhgf
48 M’XL( %\:P34  \5Z?7 <QW7G[&(!@A 8@1"_/ZP10%*R1.*;) C+UH( "%("
49  <FILE TRANSFER TRUCATED FOR EXAMPLE>
50 1 ?C^$O\%\+\!/$3,:!)$   ’
```

Command Hiding and Output Hiding

```
51
52 end
53 # exit
54 pascal> logout
```

Figure 11-2: Transcript of a Stealthy Eject Attack

In lines 34 through 54 the attacker has already gained root access and is now performing some malicious actions. The attacker uses both command hiding with glob constructs and output hiding using both uuencode and gzip. For example, on line 46 the attacker types "# /bi?/c?t /et?/p?s?w? | /u?r/l?c?l/b?n/g?ip | /b?n/u?e?c?d? hghhgf", which looks like garbage, but is interpreted by the shell as "# /bin/cat /etc/passwd | /usr/local/bin/gzip | /bin/uuencode hghhgf". Each of these techniques makes the attack harder to find for an intrusion detection system that is depending on recognition of key words for attack detection.

The User to Root attacks that were inserted into the data for the 1998 DARPA intrusion detection evaluation contained a mix of both stealthy and clear User to Root Attacks. For User to Root attacks in the four weeks of test data, 50 percent were performed in the clear with no actions, 25 percent were performed in the clear with actions, and 25 were performed stealthily with stealthy actions.

The stealthy methods presented in this section represent only a small subset of the possible steps an attacker could take in an attempt to make a User to Root attack less noticeable. Future evaluations will need to explore these additional methods more completely. As each generation of intrusion detection system learns how to defeat new stealthy methods, attackers will likely develop newer and better stealthy techniques.

11.4 Avoiding Detection of Remote to Local (R-?-L) Attacks

Many Remote to Local attacks have a unique signature that is hard to alter. In a dictionary-based password guessing attack the user simply must connect to a service many times and try different passwords. If an attacker wants to gain access to a machine by remotely overflowing the imap service they have to transmit the code that overflows the buffer across the network in the clear. For these reasons, there are fewer options for making Remote to Local attacks stealthy than there are for User to Root attacks. However, once an attacker has gained access and is interacting with a shell, all the same methods that were outlined in the User to Root section above can be used to hide further actions the attacker might perform. None of the Remote to Local User attacks in the 1998 DARPA intrusion detection evaluation were designed to be stealthy.

11.5 Actions

Many of the attacks included within the data collected for the 1998 DARPA intrusion detection evaluation include some set of actions that were not directly related to the task of either gaining access or denying service. In some cases these actions were performed as part of a scenario where an attacker had a specific goal in mind. But, in many cases even attacks that were not developed with a specific scenario in mind contain a few actions that an attacker performs after gaining access. These actions were chosen from a set that was identified by looking at transcripts of actual intrusions and by considering the goals that an attacker might have. The following are descriptions of the actions that were performed during the simulation:

- **id** An attacker uses the "id" program to check whether an exploit obtains root access
- **cat /etc/password, cat /etc/shadow** Once the attacker has collected the contents of the password file the contents can be decrypted offline.
- **netstat** Netstat provides an attacker with a list of the network services a host offers.
- **mount, showmount** An attacker runs the mount or showmount command to look for NFS mounted drives that might provide access to another machine or additional data.
- **cat /etc/hosts.equiv** The "hosts.equiv" file contains a list of trusted machines. By adding new machines to this list of trusted hosts an attacker can assure later access.

- **creation of a suid root shell** By making a copy of /bin/sh or some other shell and setting its permissions to make the executable suid root the attacker can guarantee that they will later be able to obtain root access.
- **w, who** An attacker who is trying to avoid detection can use these commands to see who else is logged in to the system.
- **editing .rhosts** The .rhosts file is similar to /etc/hosts.equiv in that it specifies a list trusted hosts. If an attacker can add the string "+ +" to /root/.rhosts the system will allow anyone to perform remote commands (rsh, rlogin, rcp, etc.) as root on that system.
- **uname** The uname command gives an attacker information about the hardware and operating system of the victim host.
- **starting up new services** Attacker can start a new listening port for later access to the machine. They can start up the tftp service that will later allow them to read any world-readable file on the system without authenticating themselves. They can bind a second listening telnet daemon to some port other than the normal port (port 23) to get around a firewall. Or, attackers can set up the netcat program to listen on a port and give them a shell, or provide them with some piece of information whenever someone connects to that port and supplies a "magic word". More examples are available in [31].

Chapter 12

Attack Planning and Data Collection

Nine weeks of data were collected during the 1998 DARPA intrusion detection evaluation. This data consisted of normal traffic as well as inserted attacks. Of these nine weeks of data, the first seven weeks are considered to be the "training" dataset and the last two weeks are the "testing" dataset. After the first seven weeks of data was collected, the data and an answer key listing the time, name, and affected machines for each attack was distributed to participants. The participants then trained their intrusion detection systems on this data. After the training phase was complete, two additional weeks of data was distributed. Participants attempted to find attacks in this two weeks of data and returned a list of found attacks to Lincoln Laboratory. These lists of found attacks were then scored to determine the participant's success rate and false alarm rate across various categories of attacks. The remainder of this chapter discusses the method used to select attacks and keep track of them, shows the actual numbers of attacks present in the 1998 evaluation, and describes the process for validating that an attack occurred.

12.1 Planning and Keeping Track of Attacks

A complete schedule of all attacks for the nine week period was developed by hand prior to the beginning of the data collection process. The number of instances of each of the 32 attack types was kept roughly equal. Once the schedule was created, the schedule was stored in electronic form and was used to create the “exs” scripts that were used to run each attack. A collection of CGI scripts was written to allow the schedule to be conveniently viewed in HTML format. The complete attack schedule for the two weeks of testing data can be found in Appendix A of this document.

12.2 Numbers of Attacks

Table 12-1 shows the number of instances of each type of attack that was inserted in the 1998 DARPA intrusion detection evaluation. Each row represents a type of attack. These attack types have been grouped by their category: User to Root, Remote to Local User, Denial of Service, or Probe. The second major column lists the number of instances of the current attack type that were performed in the clear. Within this second column the data is further divided into subcolumns listing the number of attacks of each type in the training data (first seven weeks), the testing data (last two weeks) and the total over all nine weeks of data. The third column shows the number of instances of each attack type that were performed stealthily. Note that no Remote to Local User attacks or Denial of Service attacks were stealthy. Finally, the third column aggregates the sums from the previous two columns, and shows the total number of instances (both clear and stealthy) of attacks of each type. For example, there were 46 Eject attacks in the simulation. Of these 46 Eject attacks, 10 were stealthy and 36 were performed in the clear. Of the 36 clear Eject attack instances, 29 were in the training data, and 7 were in the testing data.

Name	Attacks Performed in the Clear			Stealthy Attacks			All Attacks (Clear + Stealthy)
	Training	Testing	Total	Training	Testing	Total	Total
User to Root	**67**	**24**	***91***	**18**	**5**	***23***	***114***
Eject	29	7	*36*	8	2	*10*	*46*
Fdformat	12	7	*19*	4	3	*7*	*26*
Ffbconfig	11	2	*13*	6	0	*6*	*19*
Perl	15	1	*16*	0	0	*0*	*16*
Ps	0	4	*4*	0	0	*0*	*4*
Xterm	0	3	*3*	0	0	*0*	*3*
Remote to Local User	**19**	**15**	***34***	**0**	**0**	***0***	***34***
Dictionary	3	2	*5*	0	0	*0*	*5*
Ftp-Write	2	1	*3*	0	0	*0*	*3*
Guest	4	0	*4*	0	0	*0*	*4*
Imap	4	1	*5*	0	0	*0*	*5*
Named	0	3	*3*	0	0	*0*	*3*
Phf	6	1	*7*	0	0	*0*	*7*
Sendmail	0	3	*3*	0	0	*0*	*3*
Xlock	0	2	*2*	0	0	*0*	*2*
Xsnoop	0	2	*2*	0	0	*0*	*2*
Denial Of Service	**65**	**34**	***99***	**0**	**0**	***0***	***99***
Apache2	0	3	*3*	0	0	*0*	*3*
Back	4	2	*6*	0	0	*0*	*6*
Land	7	2	*9*	0	0	*0*	*9*
Mailbomb	0	1	*1*	0	0	*0*	*1*
Neptune	13	7	*20*	0	0	*0*	*20*
Process Table	0	3	*3*	0	0	*0*	*2*
Smurf	19	8	*27*	0	0	*0*	*27*
Syslog	4	2	*6*	0	0	*0*	*6*
Teardrop	18	4	*22*	0	0	*0*	*22*
UDPStorm	0	2	*2*	0	0	*0*	*2*
Probe/Surveillance	**50**	**14**	***64***	**0**	**0**	***0***	***64***
IPSweep	22	3	*25*	0	0	*0*	*25*
Mscan	0	1	*1*	0	0	*0*	*1*
Nmap	12	6	*18*	5	3	*0*	*18*
Saint	0	2	*2*	0	0	*0*	*2*
Satan	16	2	*18*	0	0	*0*	*18*
Total for All Attacks	**201**	**87**	***288***	**18**	**5**	***23***	***311***

Table 12-1: Instances of Non-Scenario Attacks

12.3 Verification of Attack Success

A major unanticipated difficulty of the process of creating the evaluation dataset was verifying that the inserted attacks actually ran and had the desired effect. Because the simulation network was not designed with automation of this task in mind, all verification of attack success was performed by a human operator after the data had been collected.

The verification process involved looking at logs generated by the “exs” script for each attack after it had run. This process was not ideal. In a more carefully designed verification strategy, the success or failure of an attack instance would be determined automatically at run time by looking at the actual data collected for that attack (tcpdump data, and BSM logs). Without such an automated system in place, the task of verifying attack success turned out to be quite time consuming and prone to error. A system for automatically validating attack success is being developed for use in the 1999 version of the Lincoln Laboratory DARPA intrusion detection evaluation.

Chapter 13

Results and Future Work

13.1 Results of the 1998 Evaluation

The 1998 DARPA intrusion detection evaluation was extremely successful in providing the first comprehensive realistic evaluation of intrusion detection systems. The results were useful in focusing research and highlighting the current capabilities and recent advances of existing intrusion detection systems. The procedures used to create the data and generate attacks worked well and can easily be reused to generate more data for future evaluations. Researchers who participated in the evaluation were able to see the strong and weak points of their own approaches, and the whole intrusion detection research community gained a great deal of insight about the strengths and weaknesses of current efforts in the field as a whole.

Although full discussion of the detailed results of the evaluation are outside of the scope of this thesis, there were a few broad observations that will be useful in guiding future efforts to evaluate intrusion detection systems using simulated computer attacks.

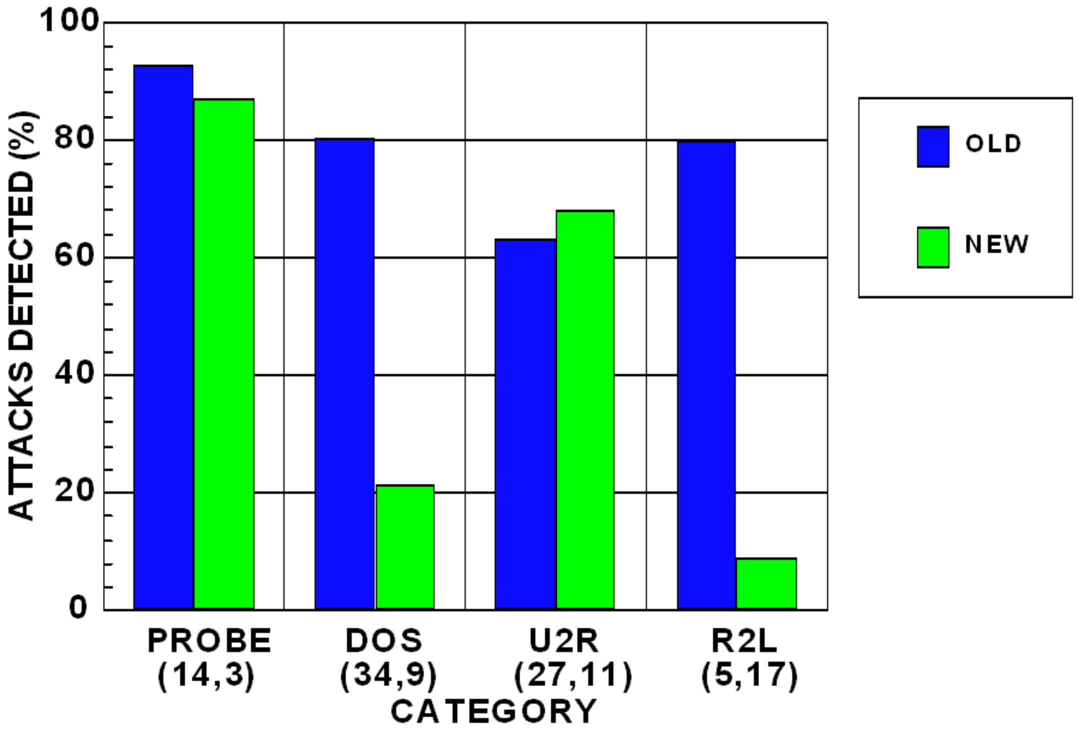


Figure 13-1: Current Intrusion Detection Systems Cannot Find New DoS and Remote to User Attacks

First, current intrusion detection research systems represent a dramatic improvement over older keyword spotting systems. The best combination of 1998 evaluation systems provides more than two orders of magnitude of reduction in false alarm rate with greatly improved detection accuracy. Second, current Intrusion Detection Systems can successfully identify older, known attacks with a low false alarm rate, but do not perform as well when identifying novel or new attacks. Figure 13-1 graphically shows this result by comparing the detection rates of the three best evaluated systems that used tcpdump data. The vertical axis of the chart represents the percentage of attacks detected when set to a threshold that allows for 10 false alarms/day. Results are broken up into the four attack categories (Probe, Denial of Service, User to Root, and Remote to Local) along the horizontal axis. This figure shows us that systems were able to

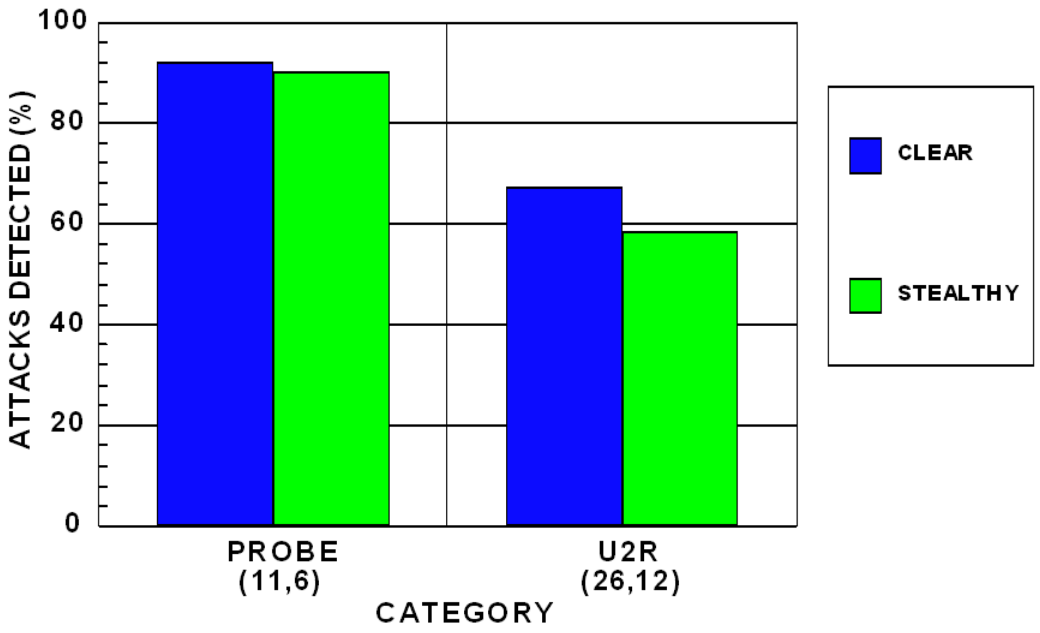


Figure 13-2: Clear vs. Stealthy Attack Detection Rates

(Top 3 Systems, TCPDump Data, 10 false alarms/day

generalize well to detect new Probe and User to Root attacks, but almost completely missed new Denial of Service and Remote to Local attacks.

Finally, the stealthy approaches presented in Chapter 11 did not significantly degrade the performance of the evaluated systems. Figure 13-3 compares the detection rate (for the top three systems at a false alarm rate of 10/day) for clear and stealthy Probes and User to Root Attacks. The rate of detection is represented on the vertical axis and Results are broken up into the two attack categories that can be made stealthy (Probes and User to Root) along the horizontal axis. Detection rates for stealthy attacks were not significantly lower than for attacks performed in the clear. Two possible reasons for the relative ineffectiveness of the stealthy attacks are the fact that many stealthy attacks were included in the training dataset, and the fact that many of the stealthy approaches discussed in Chapter 11 were designed specifically to thwart systems that rely on keyword spotting. More detailed descriptions of these results can be found at the Lincoln Laboratory Intrusion Detection Evaluation Website [42].

13.2 Future Work

13.2.1 Inclusion of Windows NT and Other Systems in the Evaluation

The 1998 Intrusion Detection evaluation focused on attacks targeted at UNIX servers, because at the current time these UNIX based machines are the most common server platform within the defense community. However, use of Windows NT servers is growing in both military and commercial environments, and attacks against Windows NT servers should be included in future evaluations. As the landscape of computer networks continues to evolve, the evaluation must evolve and adapt in order to maintain its relevance and usefulness.

13.2.2 Better Automation of Attack Planning and Verification

In the 1998 evaluation, all attack planning and verification was performed by hand. These tasks proved to be very time intensive and hindered the timely completion of the evaluation. In the future, this process should be automated. All that should be required of the human operator is to supply a database of attacks and their variants. Once this database has been collected, an automated system should be able to plan out a schedule, and generate the “exs” scripts for all of the attack instances in the evaluation. After the “exs” scripts have been run, automatic verification routines should be developed to look at the collected data and validate that the attacks occurred successfully. Developing these automated tools will take a great deal of effort, but once they have been completed the process of conducting future evaluations will be greatly simplified.

13.2.3 Improved Stealthy Attacks

The results presented in section 13.1 show that the stealthy methods used in the 1998 DARPA evaluation were largely unsuccessful. Stealthy attacks were ineffective because (1) Examples of stealthy attacks were provided in the training data and many intrusion detection systems were trained to find stealthy actions instead of the actual break-in, (2) Only a small part of the development effort focussed on making attacks stealthy, and (3) All of the user to root attacks created a root shell, which makes them relatively easy to detect. Future evaluation efforts should dedicate a large percentage of total attack development time to further developing and exploring new methods of achieving stealthiness.

Appendix A

Attack Schedule for Test Dataset

Day	Name	Time	Source	Dest	Type	Stealthy
Week 1						
Mon	portsweep	10:51:21	207.136.86.223	marx	probe	
Mon	pod	14:49:52	207.103.80.104	marx	dos	
Mon	teardrop	16:08:16	123.123.123.123	zeno	dos	
Mon	syslog	16:46:28	1.1.1.1	pascal	dos	
Tues	satan	08:31:46	153.107.252.61	marx	probe	
Tues	format	09:11:57	209.74.60.168	pascal	u2r	Y
Tues	apache2	10:11:30	207.181.92.211	marx	dos	
Tues	format2	15:44:55	194.27.251.21	pascal	u2r	
Tues	neptune	18:15:57	166.102.114.43	zeno	dos	
Tues	smurf	19:43:50	100subs	marx	dos	
Wed	httptunnel	09:32:05	197.182.91.233	pascal	scenario	
Wed	nmap	10:34:10	207.253.84.13	linux2	probe	
Wed	teardrop	16:30:05	123.123.123.123	zeno	dos	
Wed	smurf	20:07:38	199.174.194	marx	dos	
Wed	eject	21:52:01	153.10.8.174	pascal	u2r	Y
Thurs	pod	10:16:10	152.204.242.193	*	dos	
Thurs	nmap	10:23:19	207.253.84.13	linux2	probe	
Thurs	named	10:33:08	152.169.215.104	linux	r2l	
Thurs	eject	14:32:32	194.7.248.153	pascal	u2r	Y
Thurs	land	14:45:13	zeno	zeno	dos	
Thurs	smurf	15:10:48	10subs	linux3	dos	
Fri	snmpgetattack	08:03:06	207.230.54.203	loud	scenario	
Fri	teardrop	09:27:09	194.7.248.153	marx	dos	
Fri	format	10:06:37	194.7.248.153	pascal	u2r	Y
Fri	back	12:41:54	204.97.153.43	marx	dos	
Fri	neptune	14:41:28	9.9.9.9	pascal	dos	
Fri	processtable	19:20:35	Calvin	pascal	dos	
Fri	neptune	19:50:59	10.20.30.40	pascal	dos	
Week 2						
Mon	snmpgetattack	08:03:07	207.230.54.203	loud	scenario	
Mon	named	08:55:02	calvin	marx	r2l	
Mon	xlock	09:11:04	194.7.248.153	marx	r2l	
Mon	xlock2	09:27:27	194.7.248.153	pascal	R2l	
Mon	smurf	09:30:41	10subs	pascal	dos	
Mon	multihop	09:36:10	206.229.221.82	marx	scenario	
Mon	ipsweep	09:36:07	204.97.153.43	*	probe	
Mon	xsnoop	09:38:58	194.27.251.21	pascal	r2l	
Mon	named	09:54:39	204.97.153.43	linux1	r2l	
Mon	smurf	10:34:11	all.attackers	marx	dos	
Mon	saint	11:01:23	calvin	pascal	probe	
Mon	ffb	11:15:16	197.218.177.69	pascal	u2r	
Mon	portsweep	14:01:26	202.247.224.89	pascal	probe	
Mon	pod	16:11:36	207.103.80.104	pascal	dos	
Mon	apache2	18:22:34	196.227.33.189	marx	dos	
Mon	format	20:46:48	195.73.151.50	pascal	u2r	
Mon	udpstorm	21:06:52	172.16.112.50	Zeno	dos	
Mon	format2	23:10:31	202.49.244.10	Pascal	u2r	
Mon	smurf	23:16:43	152.169.215,202	Marx	dos	

			.77.162			
Mon	smurf	23:26:37	255.255.255.255 .et.al	Pascal	dos	
Tues	warezmaster	08:08:29	200.27.121.118	Pascal	scenario	
Tues	portsweep	08:27:28	135.8.60.182	Marx	dos	
Tues	multihop	09:00:19	206.229.221.82	Marx	scenario	
Tues	perlmagic	09:08:32	166.102.114.43	Marx	u2r	
Tues	xsnoop	09:09:03	200.27.121.118	Pascal	r2l	
Tues	pod	10:33:08	209.30.70.14	Linux1	dos	
Tues	dict1	11:23:37	208.239.5.230	Marx	r2l	
Tues	format1	11:55:17	209.30.71.165	Pascal	u2r	Y
Tues	sendmail	12:14:12	204.97.153.43	Marx	r2l	
Tues	eject	13:20:35	195.115.218.108	Pascal	u2r	
Tues	satan	15:17:26	208.253.77.185	Zeno	probe	
Tues	mscan	17:36:25	207.75.239.115	*	probe	
Tues	processtable	18:04:07	calvin	Pascal	Dos	
Tues	smurf	21:03:36	209.1.12.*	Linux2*	dos	
Wed	snmpgetattack	08:03:06	207.230.54.203	Loud	scenario	
Wed	warezclient	08:15:06	all.attackers	Pascal	scenario	
Wed	ps	10:06:44	209.154.98.104	Pascal	u2r	
Wed	nmap	10:16:15	207.253.84.13	Linux2	probe	
Wed	dict1	10:24:39	152.169.215.104	Marx	r2l	
Wed	xterm	10:26:12	194.27.251.21	Linux1	u2r	
Wed	rootkit	10:33:27	153.10.8.174	Marx	scenario	
Wed	xterm	10:52:58	197.182.91.233	Linux1	r2l	
Wed	neptune	11:11:28	10.140.175.80	Zeno	dos	
Wed	udpstorm	12:40:23	pascal	Zeno	dos	
Wed	loadmodule	13:32:38	205.180.112.36	Zeno	u2r	
Wed	syslog	13:46:54	194.7.248.153	Pascal	dos	
Wed	imap	14:36:15	calvin	Marx	r2l	
Wed	back	15:46:55	209.117.157.183	Marx	dos	
Wed	ps	19:26:57	197.218.177.69	Pascal	u2r	
Wed	ipsweep	20:07:16	194.7.248.153	*	probe	
Wed	ffb	20:59:25	208.239.5.230	Pascal	u2r	
Thurs	snmpgetattack	08:03:06	207.230.54.203	Loud	Scenario	
Thurs	warezclient	08:15:05	all.attackers	Pascal	Scenario	
Thurs	multihop	09:00:09	206.229.221.82	Pascal	Scenario	
Thurs	eject	09:15:00	135.008.060.182	Pascal	u2r	
Thurs	portsweep	09:36:05	195.73.151.50	Pascal	Probe	Y
Thurs	nmap	09:41:52	207.253.84.13	Linux2	probe	
Thurs	sendmail	10:44:06	207.230.54.203	Marx	R2l	
Thurs	loadmodule	11:37:27	135.13.216.191	Zeno	U2r	
Thurs	eject	11:59:43	197.182.91.233	Pascal	U2r	
Thurs	httptunnel	12:30:03	pascal	Calvin	scenario	
Thurs	neptune	13:30:49	209.74.60.168	Zeno	Dos	
Thurs	eject2	13:42:55	197.182.91.233	Pascal	U2r	
Thurs	processtable	17:56:34	calvin	Pascal	Dos	
Thurs	neptune	20:15:14	135.13.216.191	Pascal	Dos	
Thurs	format	21:13:12	206.186.80.111	Pascal	U2r	
Thurs	mailbomb	21:14:46	204.233.47.21	Pascal	Dos	
Thurs	nmap	23:26:16	128.223.199.68	Zeno	Probe	
Fri	snmpgetattack	08:04:11	207.230.54.203	Loud	Scenario	
Fri	warezclient	08:16:11	all.attackers	Pascal	Scenario	
Fri	xterm	08:43:22	calvin,209.167.99.71	Linux1	u2r	
Fri	ftp-write	08:49:26	194.27.251.21	Pascal	R2l	
Fri	ps	08:53:05	207.136.86.223	Pascal	U2r	
Fri	httptunnel1	09:01:08	pascal	Mars	scenario	
Fri	teardrop	09:33:40	222.222.222.222	Marx	Dos	
Fri	saint	10:20:15	197.218.177.69	*	probe	

Fri	neptune	10:30:11	18.28.38.48	Pascal	Dos	
Fri	ps2	12:27:21	202.49.244.10	Pascal	u2r	
Fri	eject	12:48:24	153.107.252.61	Pascal	u2r	
Fri	land	13:27:39	Zeno	Zeno	u2r	
Fri	sendmail	16:22:02	194.7.248.153	Marx	R2l	
Fri	ipsweep	16:46:15	135.13.216.191	*	Probe	
Fri	pod	18:16:42	196.227.33.189	linux1	Dos	
Fri	phf	20:18:38	197.182.91.233	Marx	r2l	
Fri	apache2	21:04:50	202.72.1.77	Marx	dos	

References

[1] Anderson, D., T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. "Safeguard Final Report: Detecting Unusual Program Behavior Using the NIDES Statistical Component," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, December 1993.

[2] Anonymous. *Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network*, Chapter 15, pp. 359-362. Sams.net , 201 West 103rd Street, Indianapolis, IN, 46290. 1997.

[3] Bishop, M., S. Cheung, et al. "The Threat from the Net", IEEE Spectrum, 38(8). 1997.

[4] Bugtraq Archives (e-mail regarding Apache vulnerability). http://www.geek-girl.com/bugtraq/1998_3/0442.html. August 7, 1998.

[5] Bugtraq Archives. http://geek-girl.com/bugtraq/1997_4/0283.html. November 13, 1999.

[6] Bugtraq Archives. http://www.geek-girl.com/bugtraq/1999_1/0852.html. February 19 , 1999.

[7] CERT Advisory CA-93.10. http://www.cert.org/ftp/cert_advisories/CA-93%3a10.anonymous.FTP.activity. July 14, 1993.

[8] CERT Advisory CA-93.18,CA-95:12. http://www.cert.org/ftp/cert_advisories/CA-95:12.sun.loadmodule.vul. September 19, 1997.

[9] CERT Advisory CA-95.09. http://www.cert.org/ftp/cert_advisories/CA-95%3a09.Solaris-ps.vul. August 20, 1995.

[10] CERT Advisory CA-96.01. http://www.cert.org/ftp/cert_advisories/CA-96.01.UDP_service_denial. February 8, 1996.

[11] CERT Advisory CA-96.06. http://www.cert.org/ftp/cert_advisories/CA-96.06.cgi_example_code. March 20, 1996.

[12] CERT Advisory CA-96.12. http://www.cert.org/ftp/cert_advisories/CA-96.12.suidperl_vul. June 26, 1996.

[13] CERT Advisory CA-96.21. http://www.cert.org/ftp/cert_advisories/CA-96.21.tcp_syn_flooding. September 19, 1996.

[14] CERT Advisory CA-96.26. http://www.cert.org/ftp/cert_advisories/CA-96.26.ping. December 16, 1996.

[15] CERT Advisory CA-97.05. http://www.cert.org/ftp/cert_advisories/CA-97.05.sendmail. January 28, 1997.

[16] CERT Advisory CA-97.09. http://www.cert.org/ftp/cert_advisories/CA-97.09.imap_pop. April 7, 1997.

[17] CERT Advisory CA-97.28. http://www.cert.org/ftp/cert_advisories/CA-97.28.Teardrop_Land. December 16, 1997.

[18] CERT Advisory CA-98.01. http://www.cert.org/ftp/cert_advisories/CA-98.01.smurf. January 5, 1998.

[19] CERT Advisory CA-98.05. http://www.cert.org/ftp/cert_advisories/CA-98.05.bind_problems. April 8, 1998.

[20] CERT Incident Note. http://www.cert.org/incident_notes/IN-98.02.html. July 2, 1998.

[21] CERT Vulnerability Note VN-98.01. http://www.cert.org/vul_notes/VN-98.01.XFree86.html. May 3, 1998.

[22] Computer Emergency Response Team Website. http://www.cert.org.

[23] Cisco Systems, Inc. "NetRanger Intrusion Detection System Technical Overview," http://www.cisco.com/warp/public/778/security/netranger/ntran_tc.htm

[24] COAST FTP Site. ftp://coast.cs.purdue.edu/pub/tools/unix/satan/. 1998

[25] Cunningham, R. K., R. P. Lippmann, D. J. Fried, S.L. Garfinkel, I. Graf, K. R. Kendall, S.E. Webster, D. Wyschogrod, and M A. Zissman, (1999) "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation," In Proceedings ID'99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute.

[26] S. Durst, T. Champion. Packet Address Swapping for Network Simulation. Patent application, Air Force Research Laboratory. March 1999.

[27] Simson Garfinkel and Gene Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, 2nd edition, April 1996.

[28] Heberlein, T. "Network Security Monitor (NSM) – Final Report", U.C. Davis: February 1995, http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf.

[29] L. Todd Heiberlein, Gihan V. Dias, Karl N. Levit, Biswanath Mukherjee, Jeff Wood, and Divid Wolber. A network security monitor. In *Proceedings of the 1990 Symposium on Research in Security and Privacy*, pages 296-304, Oakland, CA, May 1990. IEEE.

[30] Heberlein, T.L., G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. "A Network Security Monitor", in 19990 IEEE Symposium on Research in Security and Privacy. pp. 296-304.

[31] *hobbit* hobbit@avian.org. Netcat README file. http://www.l0pht.com/~weld/netcat/readme.html. 1998.

[32] D. Icove, K. Seger, W. VonStorch. *Computer Crime: A Crimefighter's Handbook*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, August 1995.

[33] Koral Ilgun. USTAT : A real-time intrusion detection system for UNIX. Master's thesis, Univers iry of California Santa Barbara, November 1992.

[34] Impack binaries on Rootshell.com. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199804/impack103.tar.gz.html. April 13, 1998.

[35] Internet Security Systems X-Force. http://www.iss.net.

[36] R. Jagannathan, Teresa Lunt, Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalai, Hal Havitz, Peter Neumann, Ann Tarnaru, and Alfonso Valdes. System design document: Next-generation intrusion detection expert system (NIDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, March 1993.

[37] Javitz, H.S. and A. Valdes, "The NIDES Statistical Component Description and Justification," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, March 1994.

[38] Kemmerer, Richard A. "NSTAT: A Model-based Real-time Network Intrusion Detection System," Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, http://www.cs.ucsb.edu/TRs/TRCS97-18.html

[39] Ko, C., M. Ruschitzka, and K. Levitt. "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specifications-Based Approach," In Proceedings 1997 IEEE Symposium on Security and Privacy, pp. 134-144, Oakland, CA: IEEE Computer Society Press.

[40]Lawrence Berkeley National Laboratory, Network Research Group Homepage. http://www-nrg.ee.lbl.gov/. May 1999.

[41] Lawrence Livermore National Laboratory. "Network Intrusion Detector (NID) Overview," Computer Security Technology Center, http://ciac.llnl.gov/cstc/nid/intro.html.

[42] Lincoln Laboratory ID Evaluation Website, MIT, http://www.ll.mit.edu/IST/ideval/index.html. 1999.

[43] Lippmann, R.P., Cunningham, R.K., Webster, S.E., Graf, I., Fried, D. Using Bottleneck Verification to Find Novel New Computer Attacks with a Low False Alarm Rate. Unpublished Technical Report. 1999.

[44] Lunt, T.F. "Automated Audit Trail Analysis and Intrusion Detection: A Survey", in Proceedings 11th National Computer Security Conference., pp. 65-73. 1988.

[45] NMAP Homepage. http://www.insecure.org/nmap/index.html. 1998

[46] Parras, Phillip A. and Peter G. Neumann. "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances," In Proceedings 20th National Information Systems Security Conference, Oct 7, 1997.

[47] Paxon, Vern. "Bro: A System for Detecting Network Intruders in Real-Time," In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, http://www.aciri.org/vern/paper.html.

[48] Nicholas Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, Ronald Olsson. A Methodology for Testing Intrusion Detection Systems. Technical report, University of California, Davis, Department of Computer Science, Davis, CA 95616, September 1995.

[49] *Real Secure 2.5 User Manual*, Chapter 6. Internet Security Systems, Atlanta, GA. [available at http://download.iss.net/manuals/rs25.tar.Z]

[50] Rootshell Website. http://www.rootshell.com. 1999.

[51] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/perl-ex.sh.html. August 26, 1996.

[52] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/solaris_ps.txt.html. June 11, 1997.

[53] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/fdformat-ex.c.html. March 24, 1997.

[54] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199711/sol_syslog.txt.html. November 6, 1997.

[55] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199801/beck.tar.gz.html. Jan 1, 1998.

[56] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199805/xterm_exp.c.html. June 4, 1998.

[57] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199806/mscan.tgz.html. June 24, 1998.

[58] Saint Website. http://www.wwdsi.com/saint. 1998.

[59] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, K. Levitt, C. Wee, R. Yip, D. Zerkle, and J. Hoagland. GrIDS—a graph based intrusion detection system for large networks. 1996. [available at http://seclab.cs.ucdavis.edu/arpa/grids/welcome.html]

[60] Sun Microsystems Security Bulletin: #00138. http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/138. 17 April, 1997.

[61] Sun Microsystems Security Bulletin: #00140. http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/140. 14 May, 1997.

[62] Sun Microsystems, Solaris Security Website. http://www.sun.com/solaris/2.6/ds-security.html. May 1999.

[63] Sundaram, A. “In Introduction to Intrusion Detection”, Crossroads: The ACM Student Magazine, 2(4). 1996.

[64] Daniel Weber. A Taxonomy of Computer Intrusions. Master’s thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

[65] Seth Webster. The Development and Analysis of Intrusion Detection Algorithms. Master’s Thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.