ABSTRACT

EVALUATION OF INTRUSION DETECTION SYSTEMS WITH AUTOMATIC TRAFFIC GENERATION PROGRAMS

by
Friday Bassey Akpan

In this master's thesis work, a program was developed using the Perl programming language to enable user defined attack programs to run automatically. A similar program was also developed for background traffic. With this program, the different features of the Nmap exploration and scanning tool were exploited to build scenarios of attacks.

Automated scenarios of attacks running in to the order of hundreds were developed. Also, different sets of automated stealthy attacks scenarios running in to the order of hundreds were developed using the timing modes, stealthy scans and scan delay features of Nmap.

These automated attacks scenarios were employed in the evaluation of the Snort intrusion detection system. It was discovered that 73% of all the Nmap's scanning types and discovery methods that were used in this work resulted in scanning activity. The Snort intrusion detection system detected and produced alerts on every of the 73% Nmap's scan types and discovery method that resulted in scanning activity. Snort was found to have a non-existent false alarm rate and a very high detection rate of 100% using these attacks scenarios and background traffic.

The developed attacks scenarios program were found to be easy to use, efficient, and easy to expand by setting only the type of attacks, parameters of the attack, and the delay time between two successive attacks in a configuration file.

Blank Page

EVALUATION OF INTRUSION DETECTION SYSTEMS WITH AUTOMATIC TRAFFIC GENERATION PROGRAMS

by
Friday Bassey Akpan

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Engineering

Department of Electrical and Computer Engineering

January 2003

APPROVAL PAGE

EVALUATION OF INTRUSION DETECTION SYSTEMS WITH AUTOMATIC TRAFFIC GENERATION PROGRAMS

Friday Bassey Akpan

Dr. Constantine Manikopoulos, Thesis Advisor — Date
Associate Professor of Electrical and Computer Engineering, NJIT

Dr. Bin He, Committee Member — Date
System Engineer, XPRT Solutions, Inc. New Jersey

Dr. Sotirios Ziavras, Committee Member — Date
Professor and Associate Chair of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Friday Bassey Akpan

Degree: Master of Science

Date: January 2003

Undergraduate and Graduate Education:

- Master of Science in Computer Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2003

- Bachelor of Engineering in Electrical and Electronic Engineering
  (option in Communications Engineering),
  Federal University of Technology, Owerri, Nigeria, 1996

Major: Computer Engineering

My master's thesis is dedicated to the memory of my beloved brother,

Joel Bassey Akpan who left this world to be with

the Lord on November 20, 2002.

ACKNOWLEDGMENT

I am greatly indebted to my thesis advisor, Dr. Constantine Manikopoulos, who not only provided directions and supervision throughout this work, but also provided all the support and encouragement I needed. I also want to thank Dr. Bin He for the numerous contributions he made towards the successful completion of this work, and Dr. Sotirios Ziavras, my graduate advisor, for his contributions and active participation in my master's thesis.

I will not fail to mention the contributions and support of my fellow graduate students whose assistance in clarifying many unfamiliar concepts, cooperation and understanding made this work possible.

I also wish to acknowledge the contributions of Ms. Clarisa Gonzalez-Lenahan and Dr. Kane, both of the Graduate Studies Office, for the successful writing of my thesis report. They organized a master's thesis writing workshop during which they took the time to explain to me and other colleagues the report writing format and review process. Ms. Gonzalez provided guidance throughout the review process and made a number of corrections in the initial drafts of this report.

I must also say a special thank you to Jeanette Petford of the Bursar's department at NJIT, and Alexia Jones, my career counselor, of the Career Development Services at NJIT. Yolanda Agront and Mike Murphy, both of Lucent Technologies, Bell Labs Innovations, USA were also of great help to me. George Asish, David Yu and Peter Allen were friends indeed. These persons and many others so numerous to mention contributed in different ways to the successful completion of my study at the New Jersey Institute of Technology (NJIT), USA.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS

OS:	Operating System
UDP:	User Datagram Protocol
IP:	Internet Protocol
ICMP:	Internet Control Message Protocol
ACK:	Acknowledgment
RPC:	Remote Procedure Calling
FTP:	File Transfer Protocol
HTTP:	Hyper Text Transfer Protocol
TCP:	Transmission Control Protocol
PPP:	Point-to-Point Protocol
SLIP:	Serial Line Internet Protocol
CGI:	Common Gateway Interface
SNMP:	Simple Network Management Protocol
SSL/TLS:	Secure Socket Layer/Transport Layer Security
SET:	Secure Electronic Transaction
S/MIME:	Secure Multipurpose Internet Mail Extensions
PGP:	Pretty Good Privacy

CHAPTER 1

INTRODUCTION

This report is an account of a master's thesis work aimed at evaluating the Snort intrusion detection system with automatic traffic generation programs.

In order to have a number of attack programs, a program was developed using the Perl programming language. This program enables user defined attack programs to run automatically. With this program scenarios of attacks running in to the order of hundreds were developed. Similarly, scenarios of stealthy attacks running into the order of hundreds were also generated using the stealth scan types, timing modes, and scan delay features of Nmap. A number of manually administered attack programs were also used in this work.

A similar scenario program was also developed using the Perl programming language in order to have a number of background traffic necessary for the evaluation of the intrusion detection system.

The Snort intrusion detection system was evaluated using the automated attack scenarios and background traffic. The parameters that were measured are the detection rate and false alarm rate of Snort. The automated scenarios of attacks and background traffic, and other manually administered attack application programs were employed to measure these parameters for the system. The receiver operating characteristic (ROC) of Snort was also investigated.

Fusion system is discussed as an area for further work. Some approaches that have been used in designing fusion systems are also presented. Fusion systems aim at combining scenarios of alerts produced from different intrusion detection systems sensors, and report these scenarios of alerts at a single easily monitored location.

The remaining part of this report is organized as follows. In Chapter 1, the scope of this project work and the organization of this report are presented. In Chapter 2, computer and network attacks and countermeasures are discussed. In Chapter 3, the layout of the intrusion detection system network and the different programs that were used in this work are presented. In Chapter 4, a concise account of Snort and some measured quantities are given. In Chapter 5, the attack programs that were used in this work are discussed. In Chapter 6, the attack scenarios planning and development process is outlined. In Chapter 7, the background traffic planning and development process is presented. In Chapter 8, approaches adopted in the design of data fusion systems are discussed. In Chapter 9, the process of conducting the attacks, running the background traffic and collecting all the necessary data is outlined. In Chapter 10, the attack results are presented and discussed, conclusions are drawn, and areas where further work is necessary are presented. Appendices and References are presented after Chapter 10.

CHAPTER 2

COMPUTER AND NETWORK ATTACKS AND COUNTERMEASURES

The proliferation of the Internet and the application of computers and computer networks in many and diverse sensitive transactions have resulted in great security concerns given all the possible attacks on the security of a computer system or network. Computer attacks and their classes are discussed in [19][2][24-25]. These attacks could result in the interruption of services or systems, or complete denial of service, interception of data and/or modification of data.

Devising and effecting countermeasures against attacks on computers and networks is an ongoing task that requires a great deal of effort. Countermeasures against attacks on computers and networks have evolved over the years to include cryptographic algorithms and protocols underlying network security applications (like encryption, hash functions, digital signatures, and key exchange), network security tools and applications (like Kerberos, X.509v3 certificates, PGP, S/MIME, IP Security, SSL/TLS, SET, and SNMPv3), and system-level security issues (like the threats of and countermeasures for intruders and viruses, and the use of firewalls and trusted systems). Cryptography, Network Security Applications and System Security are discussed in [14].

Intrusion detection system is at the system-level of computer and network security. Intrusion detection systems detect intrusions into a network or system abuse using information the intrusion detection systems gather from the computer or network. There are quite a number of intrusion detection systems developed using different approaches. Some of these systems are discussed in [7][20-23][26].

CHAPTER 3

THE INTRUSION DETECTION SYSTEM NETWORK AND PROGRAMS

In this Chapter, the intrusion detection system network and the different programs that were used in the course of this work are discussed.

3.1 The Network

The intrusion detection system network at the New Jersey Institute of Technology (NJIT) is built primarily for project work on intrusion detection systems. The network is independent of the NJIT campus network, and it is not connected to the Internet. A detailed layout of the network is shown in Figure 3.1 below.

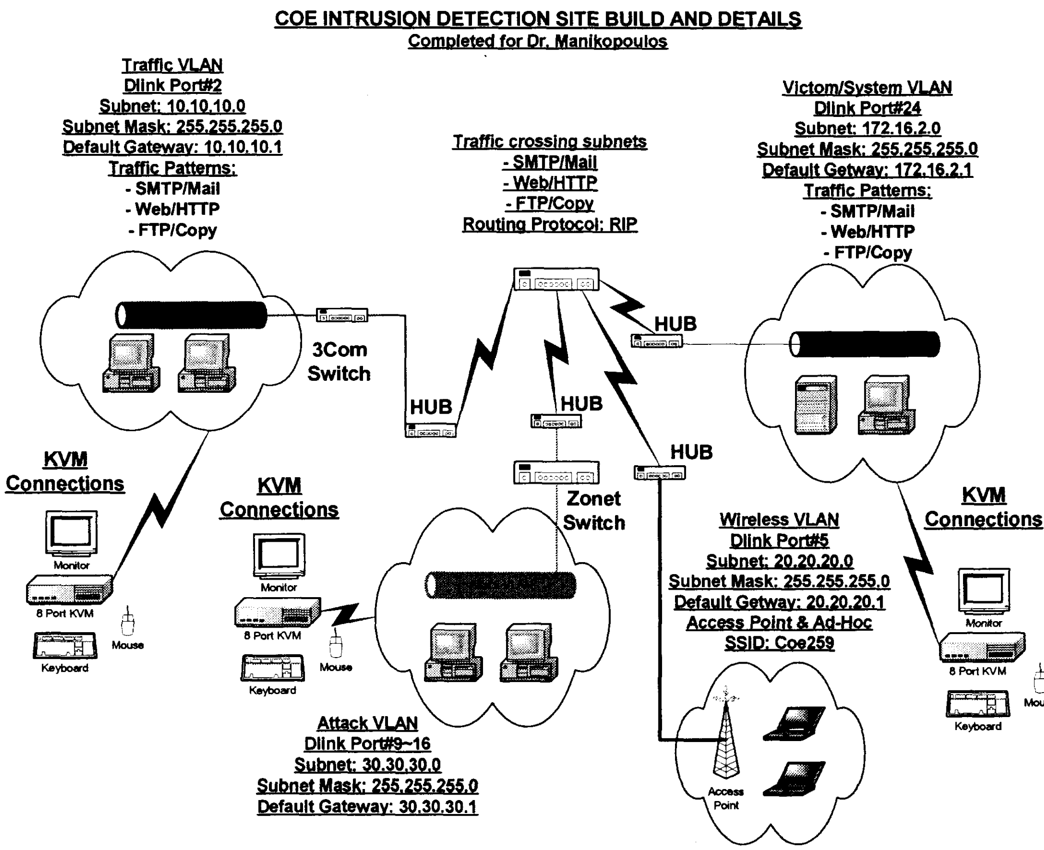


Figure 3.1 COE intrusion detection site build and details.

The network layout as shown in Figure 3.1 above comprises four network segments. The division of each of these networks terminates into a layer 2/3 switch. These switches provide the routing functionality needed for communication from one network segment to another. The network comprises four network segments with the following address spaces:

(1) Traffic VLAN #1 with address space 10.10.10.0/24

(2) System/Victim VLAN #2 with address space 172.16.2.0/24

(3) Wireless VLAN #3 with address space 20.20.20.0/24

(4) Attack VLAN with address space 30.30.30.0/24

The site built is designed to enable traffic to be generated across the system/victim network from the traffic network, and the attack network performing various attacks. Each of these networks is discussed in turn in the following sections.

3.1.1 The Traffic Network – VLAN #1

The traffic segment of the intrusion detection system network was set up to allow random patterns of predetermined background traffic across the system/victim network. The protocols used for the background traffic include Hypertext Transfer Protocol (HTTP – Web), File Transfer Protocol (FTP – File Transfer), and Simple Mail Transfer Protocol (SMTP – Mail). Some client – server background traffic have also been generated using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

The network address assignment is as follows:

10.10.10.0	-	Network Address
255.255.255.0	-	Subnet Mask

10.10.10.1 - Default Gateway

10.10.10.1 ~ 10.10.10.254 - Usable Range

Table 3.1 below gives a breakdown of the assigned usable IP address range and their corresponding host names.

Table 3.1 IP Addresses and Host Names Assignment on the Traffic Network

Host Name	IP Address
WAN- HTTP-3	10.10.10.10
WAN-FTP-4	10.10.10.11
WAN-SMTP-5	10.10.10.12
GUI_CTRL_2	10.10.10.13
WUG_1	10.10.10.14
LAN-Lin1-7	10.10.10.15
Address reserved	10.10.10.16

3.1.2 The System/Victim Network –VLAN #2

The System/Victim segment of the intrusion detection system network was set up to allow receipt of random patterns of predetermined traffic from both the traffic network, and from devices residing directly within the System/Victim network. The same protocols as in the Traffic network apply for the System/Victim network. They are: Hypertext Transfer Protocol (HTTP –Web), File Transfer Protocol (FTP – File Transfer), and Simple Mail Transfer Protocol (SMTP – Mail). Some client – server background traffic have also been generated using Transmission Control Protocol (TCP) and User Datagram Protocol (UDP).

The range of IP addresses allocated to the System/Victim network is assigned as follows:

172.16.2.0 - Network Address

255.255.255.0 - Subnet Mask

172.16.2.1 - Default Gateway

172.16.2.1 ~ 172.16.2.254 - Usable Range

Table 3.2 below gives a breakdown of the assigned usable IP address range and their corresponding host names.

Table 3.2 IP Addresses and Host Names Assignment on the System/Victim Network

Host Name	IP Address
*	172.16.2.10
WAN-WIN2K	172.16.2.11
WAN-FTP-3	172.16.2.12
WAN-HTTP-4	172.16.2.13
WUG_1	172.16.2.14*
LAN-Lin1-7	172.16.2.15
Windows Sniffer	172.16.2.17
Linux Sniffer	172.16.2.18
Address reserved	172.16.2.16

* Not assigned at the moment.

3.1.3 The Wireless Network VLAN #3

The Wireless segment of the intrusion detection system network was added to allow for intrusion detection experiment with wireless technology. Generally, wireless communication can be effected by either one of two modes of operation. These modes of operation are:

(1) The infrastructure mode of operation which involves the use of an access point connected to the wired network, and

(2) The infrastructure-less mode of operation otherwise called the ad-hoc mode of operation because an access point is not involved. In this case, the hosts (Lap Tops) communicate directly on a peer-to-peer basis.

The address space allocated to the Wireless network is assigned as follows:

20.20.20.0 - Network Address

255.255.255.0	-	Subnet Mask
20.20.20.1	-	Default Gateway
20.20.20.2	-	Access Point
20.20.20.1 ~ 20.20.20.254	-	Usable Range

Table 3.3 below gives a breakdown of the assigned usable IP address range and their corresponding host names for the Wireless network.

Table 3.3 IP Addresses and Host Names Assignment on the Wireless Network

Host Name	IP Address
Laptop_1	20.20.20.4
Laptop_2	20.20.20.5

The Service Set Identifier (SSID) for the Wireless network is Coe259.

3.1.4 The Attack Network VLAN #4

The attack segment was designed to allow different types of attacks including random patterns to be launched. These attacks could take a number of forms including:

- Probing
- Denial of Service (DOS)
- Local-to-Root
- Remote-to-Local
- Any combination of the above

The network address range allocated to the Attack network is assigned as follows:

30.30.30.0	-	Network Address
255.255.255.0	-	Subnet Mask
30.30.30.1	-	Default Gateway

30.30.30.1 ~ 30.30.30.254 - Usable Range

Table 3.4 below gives a breakdown of the assigned usable IP address range and their corresponding host names.

Table 3.4 IP Addresses and Host Names Assignment on the Attack Network

Host Name	IP Address
Attack_1	30.30.30.10
Attack_2	30.30.30.11*
Attack_3	30.30.30.12*
Attack_4	30.30.30.15
Attack_5	30.30.30.14*

* Not assigned at the moment.

3.2 Network Programs

A number of network utility programs were installed in the different network segment of the intrusion detection network to allow for a number of functions including capturing of packets, collection of tcpdump files and analysis of tcpdump files. These programs include Windump and the Ethereal Network Analyzer. Both the Windump and Ethereal Network Analyzer programs were installed on the Attack and System/Victim networks. Each of these programs is discussed in turn below.

3.2.1 Windump

Windump [11] is the *tcpdump* version for the Windows operating system. It is a packet sniffer and analyzer.

The Windump program was installed on both the attack and sniffer machines, and was used to collect attack and victim *tcpdump* files on the respective machines.

3.2.2 Ethereal Network Analyzer

Ethereal [12] is a network protocol analyzer for both Unix and Windows operating systems. It has the capability to analyze data from a live network or captured data on file on a disk. With Ethereal, summary and detailed information for each packet can be examined. The Ethereal Network Analyzer program has a number of features including a display filter that allows users to select display preferences.

The Ethereal Network Analyzer program was installed on both the attack and sniffer systems, and was used to examine and analyze *tcpdump* files recorded both at the attack and victim machines during the different test sessions.

CHAPTER 4

SNORT INTRUSION DETECTION SYSTEMS AND MEASURED QUANTITIES

Intrusion detection systems have become an important part of many network security architectures. Generally speaking, intrusion detection systems monitor the networks on which they are deployed for suspicious activity or predetermined patterns. Most network intrusion detection systems are equipped with the capability of alerting and logging these information, while a very few have real-time capability of taking corrective measures.

In this work, one network intrusion detection system, Snort was considered. A detailed discussion of this system is presented next.

4.1 Snort

Snort [7] is a packet sniffer and logger with real time alerting capability. It is rules-based, can perform content pattern matching, and detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and much more, and has been used as an intrusion detection system. Snort performs payload inspection and can filter traffic depending on the given command line instructions.

Snort is made up of three primary subsystems. These subsystems are: the packet decoder, the detection engine, and the logging and alerting systems. The decode engine is built around the supported data-link and TCP/IP protocol definitions. The supported data-link protocols are Ethernet, SLIP, and raw (PPP). The detection engine maintains Snort's detection rules in a two dimensional linked list, termed Chain Headers and Chain Options.

Snort's logging and alerting subsystems options are selected at run time with command line switches. Figure 4.1 below is an example of Snort's alert output.

```
[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/05-09:10:19.164975 30.30.30.10:3074 -> 172.16.2.0:8080
TCP TTL:127 TOS:0x0 ID:64770 IpLen:20 DgmLen:48 DF
******S* Seq: 0x33A9E5F3  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK

[**] [1:620:2] SCAN Proxy (8080) attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
12/05-09:10:19.649275 30.30.30.10:3074 -> 172.16.2.0:8080
TCP TTL:127 TOS:0x0 ID:64776 IpLen:20 DgmLen:48 DF
******S* Seq: 0x33A9E5F3  Ack: 0x0  Win: 0x4000  TcpLen: 28
TCP Options (4) => MSS: 1460 NOP NOP SackOK
```

Figure 4.1 An example of Snort's Alert output.

4.2 Performance Measures

In determining the performance of the intrusion detection system, two performance measures were employed. These are the false alarm rate and detection rate. These measures are discussed next.

4.2.1 False Alarm Rate

False alarm rate is a measure of how many alarms (false) were generated for normal traffic monitored over a specified period of time, say a day – twenty-four hours period. Background traffic of different types was employed in determining this performance measure for the Snort intrusion detection system.

4.2.2 Detection Rate

Detection rate measures the percentage of all attacks detected by a particular intrusion detection system. To determine this performance measure for Snort, a number of attacks types were used.

4.2.3 Receiver Operating Characteristic (ROC)

The Receiver Operating Characteristic (ROC) curve is a plot of the percentage correct attacks detected by an intrusion detection system versus the number of false alarms per day produced by the system. This performance measure provides a means of evaluating the trade off between the detection rate and the false alarm rate. Low false alarm rates with high detection rates means that the detection output can be relied upon. Conversely, relatively high false alarm rate with low detection rate means that the detection output can not be believed, and much more work will be needed as security analysts would be spending more hours dismissing false alarms.

CHAPTER 5

ATTACK PROGRAMS

A number of attack programs were considered in the course of this work. Some of these attack programs were automated by utilizing them in building scripts of attack scenarios using the Perl programming language. Some other types of attack programs that could not be automated due to their nature were administered manually. These attack programs are discussed in the following sections.

5.1 Nmap

Nmap [10] ("Network Mapper") is one of the attack programs that were utilized in building scripts of attack scenarios using the Perl programming language. The Nmap program is designed to scan large networks rapidly. It employs raw IP packets to determine information on what hosts are available on the network, what services (ports) they are offering, what type of packet filters/firewalls are in use, what operating system (and OS version) they are running, and other characteristics.

Nmap program provides a number of scan types, discovery methods, and options. The scan types provided by Nmap include: TCP connect() scan, TCP SYN scan, Stealth FIN, Xmas Tree, or Null scan modes, Ping scanning, UDP scans, IP protocol scans, ACK scan, Window Scan, RPC scan and List scan. The discovery methods provided by Nmap include: using TCP Ping, using TCP + ICMP, using ICMP Ping, or using the Don't Ping method. The different options provided by the Nmap program include: Fragmentation, Get Identification Information, Resolve All, Don't Resolve, Fast Scan, Operating System (OS) Detection, Random Host, and Resume. The program also provides variation in timing. The timing modes include Paranoid, Sneaky, Polite, Normal, Aggressive, and Insane.

Each of these scan types, discovery methods, options and variations in timing are discussed below.

5.1.1 Nmap Scan Modes

In this section, a concise description of the different scan types is presented.

5.1.1.1 SYN. SYN scan works by sending SYN packet as though a real connection was to be opened, and then waits for a response. A SYN/ACK is indicative that the port is listening while a RST is indicative of a non-listener. Where a SYN/ACK is received, a RST is sent to tear down the connection.

5.1.1.2 FIN. This Nmap scan type uses a bare FIN packet as the probe. The idea is to make scanning as clandestine as possible.

5.1.1.3 PING SWEEP. This Nmap scan type is useful in cases where all the information that is needed is which hosts on a network are up. It uses ICMP echo requests packets.

5.1.1.4 UDP SCAN. When performing UDP scans, Nmap sends 0 byte udp packets to each port on the target machine(s) to determine which UDP ports are open on the host(s).

5.1.1.5 NULL SCAN. Null scan has the same objective as FIN scan. It turns off all the flags.

5.1.1.6 XMAS TREE. The Xmas Tree scan type has the same objective as FIN scan, but it turns on the FIN, URG, and PUSH flags on.

5.1.1.7 IP PROTOCOL SCAN. The IP protocol scans in Nmap sends raw IP packets without any further protocol header to each specified protocol on the target machine in order to determine which IP protocols are supported on a host. Whether or not a particular protocol is used is determined by the kind of message that is received. An ICMP protocol unreachable message means the protocol is not in use.

5.1.1.8 ACK SCAN. ACK scan is usually used to map out firewall rulesets. It works by sending an ACK packet to specified ports and waiting for RSTs. If a RST comes back, then the port is not filtered, but if nothing comes back (or if an ICMP unreachable is received), then the port is filtered.

5.1.1.9 WINDOW SCAN. This type of Nmap scan is similar to ACK scan, and can detect open ports as well as filtered and non-filtered ports.

5.1.1.10 RCP SCAN. RCP scan determines which ports are RCP ports by flooding all TCP/UDP ports that have been found open with SunRPC program null commands.

5.1.1.11 LIST SCAN. List scan in Nmap generates and prints a list of IP addresses and names. In doing this, Nmap does not perform any pinging or port scanning.

5.1.1.12 CONNECT. This Nmap scan type employs the connect() system call to open a connection to interesting ports on the machine. The connection operation will succeed if the port is listening. Otherwise it will not succeed.

5.1.2 Nmap Discovery Methods

In this section, the different Nmap discovery methods are discussed.

5.1.2.1 TCP Ping. In this Nmap discovery method, TCP "ping" is used to determine what hosts are up in a network. It employs TCP ACK packets. These packets are sent into the network or to a single host. Hosts that are up will respond with a RST.

5.1.2.2 TCP + ICMP. This discovery method combines the TCP ping and ICMP discovery methods.

5.1.2.3 ICMP Ping. This discovery method uses ICMP echo request packets and waits for the corresponding echo reply packets.

5.1.2.4 Don't Ping. This discovery method allows for the scanning of networks that do not allow ICMP echo requests (or responses), may be, through their firewall. The idea is to attempt scanning without pinging host systems.

5.1.3 Nmap General Options

In this section, the different options provided by the Nmap program are discussed.

5.1.3.1 Fragmentation. This enable Nmap to cause the requested scan to use tiny IP packets. The idea is to prevent network-protecting systems to detect the actions taken against the network.

5.1.3.2 Get Identification Information. This option enables Nmap to identify the hosts on the network.

5.1.3.3 Resolve All. This option tells Nmap to do reverse DNS resolution on the active IP addresses it finds.

5.1.3.4 Don't Resolve. This option tells Nmap never to do reverse DNS resolution. The objective here may be to speed up the scanning process.

5.1.3.5 Fast Scan. This causes Nmap to scan only those ports listed in the services file that comes with Nmap.

5.1.3.6 Operating System (OS) Detection. This option enables Nmap to use a number of techniques to detect useful information about the underlying system being scanned. This information is then used to create a 'fingerprint' which it compares with its database of known OS fingerprints. With this, it proceeds to guess the operating system.

5.1.3.7 Random Host. This option tells Nmap to shuffle a group of up to 2048 hosts before it scans them.

5.1.3.8 Resume. This option enables Nmap scanning session that is cancelled due to control-C, network outage, etc. to be resumed.

5.1.4 Nmap Timing Modes

Nmap has a number of timing modes designed to meet the objective of the user. These timing modes are discussed below. Also, the particular timing modes that were used in developing the stealth attacks scenarios are mentioned in the respective sections.

5.1.4.1 Paranoid. This Nmap timing mode scans very slowly in the hope of avoiding detection by intrusion detection systems. It serializes all scans. In other words, under this timing mode, Nmap does not perform parallel scanning. Also, in this mode, Nmap generally waits at least five (5) minutes between sending packets.

The command option for this timing mode is "0." This is shown in the sce_exp.conf file in Appendix H – stealth attacks scenarios configuration files. This timing mode was used for the first eighty five (85) attacks scenarios, with large scanning time delay between individual attacks scenarios.

5.1.4.2 Sneaky. This Nmap timing mode is similar to Paranoid. The only difference here is that the waiting time between sending packets is 15 seconds (not 5 minutes).

The command option for this timing mode is "1." This timing mode was used for the eighty sixth to one hundred and seventieth (86-170) attacks scenarios after the paranoid mode, and also with somewhat large scanning delay time (lesser than for 1 - 85) between individual attacks scenarios. The part of the configuration file, sce_exp.conf file, corresponding to the stealth attacks scenarios developed using this timing mode cannot be shown due to space constraint.

5.1.4.3 Polite. This Nmap timing option is designed to ease load on the network and reduces the chances of crashing machines. It serializes the probes and waits at least 0.4 seconds between the packets being sent.

The command option for this timing mode is "2." This timing mode was used for the remaining attacks scenarios, and with even lesser scanning delay time between individual attacks scenarios. The part of the configuration file, sce_exp.conf file, corresponding to the stealth attacks scenarios developed using this timing mode cannot be shown due to space constraint.

5.1.4.4 Normal. This Nmap timing option causes the Nmap program to run as fast as possible without overloading the network or missing hosts/ports. This is the default Nmap behavior.

The option for this timing mode is "3." Many of the attacks scenarios in the normal attacks scenarios script as shown in Appendix F employed this timing mode.

5.1.4.5 Aggressive. This Nmap timing mode adds a 5 minutes timeout per host. Under this mode, Nmap does not wait more than 1.25 seconds for probe responses. This timing mode was used in the normal attacks scenarios script shown in Appendix F. The command option for this timing mode is "4."

5.1.4.6 Insane. This Nmap timing mode is only suitable for very fast networks or where capturing every piece of information is really not important. This mode times out in 75 seconds and only waits 0.3 seconds for individual probes. The command option for this timing mode is "5."

An example of Nmap's output is shown in Figure 6.1 below.

```
Nmap (V. 3.00) scan initiated Wed Dec 04 01:10:20 2002 as: nmap –sS –P0 –O –T 3 –
oN c:\temp\Friday\synn 172.16.2.1/24
Interesting ports on  (172.16.2.0):
(The 1599 ports scanned but not shown below are in state: closed)
Port      State      Service
23/tcp    open       telnet
80/tcp    open       http
Remote operating system guess: Cisco VPN 3000 or 3COM 4924 GigE Switch
Uptime 48.238 days (since Wed Oct 16 20:27:20 2002)
Interesting ports on  (172.16.2.1):
(The 1599 ports scanned but not shown below are in state: closed)
Port      State      Service
23/tcp    open       telnet
80/tcp    open       http
Remote operating system guess: Cisco VPN 3000 or 3COM 4924 GigE Switch
Uptime 48.238 days (since Wed Oct 16 20:27:20 2002)
```

Figure 5.1 A typical Nmap program output

5.2 Netcat Program

Netcat, or "nc" [9] as the actual program is named, is another form of attack program. However, it was not used in this work. Netcat can create almost any kind of connection that one would need. It can be used directly or easily driven by other programs and scripts.

In its simplest usage, Netcat creates a TCP connection to the given port on the given target host. The standard input is then sent to the host, and anything that comes back across the connection is sent to the standard output.

5.3 Other Attack Programs

Many other kinds of attack programs were also used in this work. These are presented in Table 5.1 below.

Table 5.1 Manual Attack Application Programs

Attack Name	Attack Sub-Name	Attack Other Name
Denial of service		
Battlepong		
Bloodlust		
Divine	RETRiBUTION	FyRE
		BRiMSTONE
	PLAGUEz	FLOODz
		LOCUSTs
	OMENz	HELLFyRE
		DEMONz
	INCANTATIONz	HAiLSTORM
		WHyRLWiND
Elite	anarchy	
	bomber	
	connections	
	events	
	finder	
	flood	
	nuke	
	ping	
	pingflood	
	portscan	
	resolve	
	teardrop	
	whois	
IP Spoof	dc_is	
Iping32	iping32	
Packetbuild	PckBuilder	
Ping-G	PIN-G	
Remos	RemOS	
Rocket10	Rocketv1_0	
Vai-te-ja-icmpbomber	Vai-te Ja ICMP Bomber	
WinNuke	WinNUKE	
Winsmurf	WinSmurf	
Clientttriono		
Server		
FunlinApocalypse	bmb2	
IPScanMaster	IP Scan Master	
Sscan205-scanner	Ez Converter Plus	

CHAPTER 6

ATTACK SCENARIOS

A number of attack scenarios running into the order of hundreds were developed in the course of this work for use in the evaluation of the Snort intrusion detection system. The development of these attack scenarios represents efforts at automating many of the attack types on the one hand, and administering these many and diverse attack types and options in a coordinated manner and much more conveniently, on the other. In this chapter, the attack scenarios planning and development processes are discussed.

6.1 Attack Scenarios Planning

The different attack scenarios were built from the different attack types under each kind of attack, different discovery methods, different attack options, and different timing options for each attack type as discussed in Chapter 5. Additionally, there were different combinations of these attacks, and the scan interval between them.

6.2 Attack Scenarios Development

The attack scenarios were developed using the Perl programming language with the ultimate purpose of being able to run user-defined attack programs automatically. Perl is portable. It can be used on many platforms including Unix, Linux and Windows.

The Perl programs were developed with the intent that generating attack scenarios be fairly easy by setting only the attack types, parameters and delay time between two attacks in a configuration file. The program is developed in two steps characterizing typical attacker behavior. The first step involves information gathering, and the second step involves some specific exploits based on the information gathered in the first step.

6.2.1 Information Gathering

The information-gathering step represents the first step in simulating a sequence of intrusion action in this work. Since an attacker often starts with discovery attacks to obtain important information about a victim's network, this first step of the program was developed solely for the purpose of gathering information about the victim's network. Such information may include: which hosts (or IP addresses) are running, what port numbers are open, and what services are running on these ports.

Two files were developed in this step. One is a configuration file, and the other is a Perl script. The configuration file, *pre_exp.conf* and the Perl script, *pre_exp.pl* are given in Appendix E.

When the Perl script is being executed, it calls the configuration file, and parse configuration information. The configuration information includes the victim network address (or address range) and which services to detect and other parameters. When the Perl script finishes scanning the victim network, it saves the result in a database called *pre_exp.data*. Each line of the database file contains the information of one running host.

A typical content of the *pre_exp.data* database for our network for the information gathering stage is shown below.

```
172.16.2.0    |23 80 : |53 : |
172.16.2.1    |23 80 : |53 : |
172.16.2.10   |25 80 : |111 : |
172.16.2.11   |21 80 : | : |
172.16.2.12   |21 25 80 : | : |
172.16.2.13   |21 25 80 : | : |
172.16.2.17   | : | : |
172.16.2.18   |21 22 : |111 : |

Test up: "nmap -g 1500 -n -sP -PS 172.16.2.1/27" on 01/06/03 18:02:25
Test TCP port: "nmap -g 1500 -n -sS -P0 <IP> -p 20-30,80,110" on 01/06/03 18:02:50
Test UDP port: "nmap -g 1500 -n -sS -P0 <IP> -p 53,111 -sU" on 01/06/03 18:02:50
```

Figure 6.1 Content of pre_exp.data database.

6.2.2 Attack Exploit

The attack exploit represents the second step in the simulation of intrusion action in this work. In this step, there are two configuration files and one Perl script. The configuration files are sce_cmd.conf and sce_exp.conf, and the Perl script is sce_exp.pl. The files in this step, both the configuration files and the Perl script, are given in Appendix F. Each of the configuration files and the Perl script are discussed in the following sections.

6.2.2.1 sce_cmd.conf. The configuration file, *sce_cmd.conf,* specifies each attack program and its parameters. The first column in this file is the entry name. The second column is the attack program name, and the other columns are its parameters' format.

New attack types are added to the program by defining its format in the *sce-cmd.conf* configuration file.

6.2.2.1 sce_exp.conf. The configuration file, sce_exp.conf, is the file where attack scenarios are specified. Each line in this file is a kind of known attack. The *sce_exp.conf* and *sce_cmd.conf* configuration files together define a particular attack to be carried out when the Perl script is executed.

For each attack to be carried out, and as defined in the *sce_cmd.conf* configuration file, there are typically three entries that are defined in this configuration file – *sce_exp.conf*. The first entry is the waiting time between two attack commands. This is represented as < n >s, < n >m or < n >h to indicate number of seconds, minutes or hours. The < n > represents a number. The second entry is the command to build the database. This command is represented by *newdata* on a separate line. The command, *newdata,* calls the configuration file *pre_exp.pl* each time it is executed to scan the network again in order the build the database. The third and final entry is the attack command line. The attack command line represents an attack command and its parameters, with the space character separating them. The first column in the command line of the *sce_exp.conf* configuration file is the same entry name of an attack specified in *sce_cmd.conf* configuration file. With this, the Perl script upon reaching a new (and unique) attack name, uses the information contained in the two configuration files to construct the attack command, and then, executes it.

6.2.3 Stealth Attacks Scenarios

Stealthiness in attacks scenarios building has to do with hiding an attack action from an individual monitoring the system or network, or from an intrusion detection system. The methods for making attacks stealthy depend on the type of attack.

The Nmap program used in this used is basically a surveillance/probing form of attack. For this form of attack, a number of methods have been identified for either hiding the fact that a probe is occurring, or hiding the identity of the attacker. One of the simplest ways to hide probing actions is to make the probe to occur slowly. For the Nmap program, the timing options – paranoid and sneaky – provide for this occurrence. These timing options were used to build a large number of attacks scenarios for the stealthy attacks script.

Another method to hide probing actions is to use scan delays. The scan delay option in Nmap specifies the amount of time Nmap must wait between probes. In addition to slowing the scan way down as to sneak under intrusion detection system's thresholds, scan delay can also be useful in reducing the load on the network. This method was also used in building the stealth attacks scenarios.

Yet another method to hide probing actions is to employ scan types that are inherently designed for this purpose. Such scan types do half-open connections or FIN scanning of a network. The Nmap program provides two of such scan modes. These are the SYN stealth and the FIN stealth scan attack types. The SYN stealth scan works by sending SYN packet as though a real connection was to be opened, and then waits for a response. A SYN/ACK is indicative that the port is listening while a RST is indicative of a non-listener. Where a SYN/ACK is received, a RST is sent to tear down the connection. The FIN stealth scan attack type on the other hand uses a bare FIN packet as the probe. Relative to the SYN stealth scan type, the idea in this case is to make scanning as clandestine as possible. The SYN stealth scan type was also used to develop some stealth attacks scenarios.

CHAPTER 7

BACKGROUND TRAFFIC

Background traffic emanating from the traffic network of the intrusion detection system network was used each time attacks were performed and data were being collected. Generally speaking, four kinds of background traffic namely file transfer protocol (FTP), telnet, hyper-text transfer protocol (HTTP), and Mail services are available for work of this nature. These four kinds of traffic are discussed in [13].

7.1 Background Traffic Scenarios Planning

The background traffic scenarios was planned around the available background traffic types - file transfer protocol (FTP), telnet, hyper-text transfer protocol (HTTP), and Mail services.

7.2 Background Traffic Scenarios Development

A background traffic-generating program was developed using the Perl programming language. This program generates user-defined background traffic automatically. The program contains three files: *bkgrd_cmd.conf*, *bkgrd_gen.conf*, and *bkgrd_gen.pl*. The file structure of this background traffic-generating program is similar to that of the attacks scenarios building program. The *bkgrd_cmd.conf* configuration file defines the format of each background traffic type, the *bkgrd_gen.conf* configuration file contains various background traffics to be generated, and *bkgrd_gen.pl* is the script reading the configuration information in *bkgrd_gen.conf* and generating the background traffic. These scripts are shown in Appendix G.

CHAPTER 8

FUSION SYSTEM

As intrusion detection systems are becoming more and more in common use in organizations, the trend is moving towards the use of multiple intrusion sensors. A consequence of this practice is that the work load of security personnel may increase as the same attacks may be detected and reported by different sensors. Additionally, while the detection rate is improved, the false alarm rate tends to increase.

Fusion systems are designed to overcome these issues by combining the alerts generated by the different sensors and reporting the results at a single location. This is an ongoing area of research work, and a number of approaches have been adopted in combining the alerts.

The algorithm described in [28] is probabilistic in nature. It determines the scenario membership of a new alert in time proportional to the number of candidate scenarios. In essence, it groups alerts that share a common cause.

In [29], an approach to construct attack scenarios by correlating alerts on the basis of prerequisites and consequences of intrusions. Based on the prerequisites and consequences of different types of attacks, the proposed approach correlates alerts by (partially) matching the consequence of some previous alerts and the prerequisites of some later ones.

The alerts obtained from the Snort intrusion detection system when the different attacks scenarios in this work were executed have been made available for the next phase of this work – building the alerts scenarios.

CHAPTER 9

CONDUCTING ATTACKS AND COLLECTING DATA

All the attack programs were conducted from the attack network. The different background traffics were sent into the network from the traffic network. The sniffer system is installed in the system/victim network from where data on the network is dumped. All the attacks were performed in the presence of background traffic.

In measuring the detection rate of Snort, all the automated attack scenarios and manually administered attack programs were used in the presence of background traffic. In determining the false alarm rate of Snort, only background traffics were used.

Appendix A shows the false alarm rate tests schedule for Snort intrusion detection system. Appendix B shows the tests schedule for the determining the detection rate of Snort using the Nmap programs. Appendix C shows the tests schedule for determining the detection rate of Snort using the manual attack application programs. Appendix D shows the tests schedule for determining the detection rate for Snort using the attack scenarios.

CHAPTER 10

RESULTS AND FUTURE WORK

In this Chapter, result from the different attack performed and the different background traffic ran independently is presented. Also, in this Chapter, a discussion of the results obtained, conclusions drawn, and areas for future work are presented.

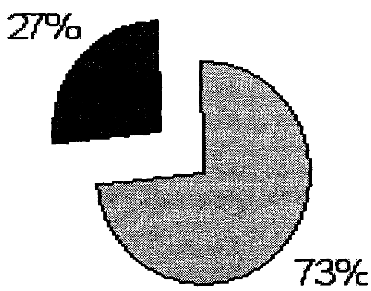
10.1 Attacks Results Summary

In the discussion of these results, reference is made to Appendices A – D showing the results and the tests schedule for all the tests that were carried out.

Table 10.1 shown below summarizes the number of Nmap attack programs that were detected by the Snort intrusion detection system. It shows similar figures for the manually administered attack application programs.

Table 10.1 Summary Result of Attacks Detected by Snort

Attack Type (And # Below)	# of Observable Scan/Attacks	# of Real Attacks	# Detected by Snort	Snort Detection Rate
Nmap Programs	48	35	35	100%



Nmap: 73% of all Scan types and Discovery Methods used resulted in Scan activity. 23% of all Scan types and Discovery Methods used did not resulted in scanning activity.

Snort: Of the 73% of Nmap Programs that resulted in scanning activity, Snort detected 100%.

Table 10.2 summarizes the number of false alarms generated by the Snort intrusion detection system when the different background traffic were present on the network.

Table 10.2 Summary Result of False Alarms Generated by Snort

Type of Traffic	Snort
FTP	0
HTTP	0
Fault Traffic	0
Client-Server (TCP)	0
Client-Server (UDP)	0

The result presented in Table 10.1 above indicates that 73% all Nmap's scan types and discovery method used in this work resulted in actual scanning of the network. Of the 73% Nmap attacks that resulted in scanning activity, the Snort intrusion detection system detected all (100%) of them.

For the false alarm rate performance, the Snort intrusion detection system produced no false alarms for the background traffic used.

The receiver operating characteristic curve for Snort intrusion detection system will show a 100% detection rate for the Nmap attack programs with 0% false alarm rate. These results are discussed below.

10.2 Discussion of Results

As seen in Appendix B, from the breakdown of all Nmap's scanning attacks performed with different discovery options, it was discovered that there were no alerts produced by the Snort intrusion detection system for any of the scan types when the ICMP discovery options is used. In the following sections, the Snort alerts collected and the scan results for a selected number of the scenarios are presented and discussed.

10.2.1 SYN Stealth with TCP Ping Discovery Option

The scan result for this scanning attack scenario is shown in Figure 10.1 below.

nmap (V. 3.00) scan initiated Sun Jan 05 20:06:08 2003 as: nmap -sS -PT -O -T 3 -oN c:\temp\friday\syntcpn --append_output 172.16.2.17
Interesting ports on LAN_SNIFF_8 (172.16.2.17):
(The 1599 ports scanned but not shown below are in state: closed)
Port State Service
135/tcp open loc-srv
139/tcp open netbios-ssn
Remote operating system guess: Windows NT4 or 95/98/98SE
Nmap run completed at Sun Jan 05 20:06:12 2003 -- 1 IP address (1 host up) scanned in 4 seconds

Figure 10.1 Result of the SYN stealth with TCP Ping Discovery option attack scenario.

Some of the Snort alert produced by this scenario is shown in Figure 10.2 below.

[**] [1:615:3] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
01/05-20:06:10.598136 30.30.30.10:55593 -> 172.16.2.17:1080
TCP TTL:43 TOS:0x0 ID:48489 IpLen:20 DgmLen:40
******S* Seq: 0x5CA55D87 Ack: 0x0 Win: 0x400 TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]

[**] [1:1227:4] X11 outbound client connection detected [**]
[Classification: Misc activity] [Priority: 3]
01/05-20:06:11.066203 172.16.2.17:6004 -> 30.30.30.10:55593
TCP TTL:128 TOS:0x0 ID:52844 IpLen:20 DgmLen:40
***A*R** Seq: 0x0 Ack: 0x5CA55D88 Win: 0x0 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS126]

Figure 10.2 Some of Snort's alerts for the SYN stealth with TCP Ping Discovery option attack scenario.

From the results shown in Figure 10.1 and Figure 10.2, it is evident that this attack scenario actually scans the network and that Snort alerts on this scanning activity.

10.2.2 SYN Stealth with TCP + ICMP Ping Discovery Option

The scan result for this scanning attack scenario is shown in Figure 10.3 below.

nmap (V. 3.00) scan initiated Sun Jan 05 20:11:33 2003 as: nmap -sS -PT -PI -O -T 3 -oN c:\temp\friday\syntcpicmpn --append_output 172.16.2.17

Insufficient responses for TCP sequencing (2), OS detection may be less accurate
Interesting ports on LAN_SNIFF_8 (172.16.2.17):
(The 1599 ports scanned but not shown below are in state: closed)
Port State Service
135/tcp open loc-srv
139/tcp open netbios-ssn
Remote operating system guess: Windows NT4 or 95/98/98SE
Nmap run completed at Sun Jan 05 20:11:38 2003 -- 1 IP address (1 host up) scanned in 5 seconds

Figure 10.3 Result of the SYN stealth with TCP + ICMP Ping Discovery option attack scenario.

Some of the Snort alert produced by this scenario is shown in Figure 10.4 below.

[**] [1:615:3] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
01/05-20:11:36.426470 30.30.30.10:49498 -> 172.16.2.17:1080
TCP TTL:36 TOS:0x0 ID:3161 IpLen:20 DgmLen:40
******S* Seq: 0x4BED69FB Ack: 0x0 Win: 0x800 TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]

[**] [1:1227:4] X11 outbound client connection detected [**]
[Classification: Misc activity] [Priority: 3]
01/05-20:11:36.482907 172.16.2.17:6005 -> 30.30.30.10:49498
TCP TTL:128 TOS:0x0 ID:61301 IpLen:20 DgmLen:40
***A*R** Seq: 0x0 Ack: 0x4BED69FC Win: 0x0 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS126]

Figure 10.4 Some of Snort's alerts for the SYN stealth with TCP + ICMP Ping Discovery option attack scenario.

From the results shown in Figure 10.3 and Figure 10.4, it is evident that this attack scenario actually scans the network and that Snort alerts on this scanning activity.

10.2.3 SYN Stealth with ICMP Ping Discovery Option

The scan result for this scanning attack scenario is shown in Figure 10.5 below.

nmap (V. 3.00) scan initiated Tue Dec 24 14:49:07 2002 as: nmap -sS -PI -T 3 -oN c:\temp\synicmpping --append_output 172.16.2.17
Nmap run completed at Tue Dec 24 14:49:37 2002 -- 1 IP address (0 hosts up) scanned in 30 seconds
Starting nmap V. 3.00 (www.insecure.org/nmap)

Note: Host seems down. If it is really up, but blocking our ping probes, try -P0
Nmap run completed -- 1 IP address (0 hosts up) scanned in 30 seconds

Figure 10.5 Result of the SYN stealth with TCP + ICMP Ping Discovery option attack scenario.
No Snort alert is produced in this case.

10.2.4 SYN Stealth with Don't Ping Discovery Option

The scan result for this scanning attack scenario is shown in Figure 10.6 below.

```
# nmap (V. 3.00) scan initiated Sun Jan 05 20:21:36 2003 as: nmap -sS -P0 -O -T 3 -oN
c:\temp\friday\syndontn --append_output 172.16.2.17
Interesting ports on LAN_SNIFF_8 (172.16.2.17):
(The 1599 ports scanned but not shown below are in state: closed)
Port       State       Service
135/tcp    open        loc-srv
139/tcp    open        netbios-ssn
Remote operating system guess: Windows NT4 or 95/98/98SE
# Nmap run completed at Sun Jan 05 20:21:40 2003 -- 1 IP address (1 host up) scanned in
4 seconds
```

Figure 10.6 Result of the SYN stealth with Don't Ping Discovery option attack scenario.

Some of the Snort alert produced by this scenario is shown in Figure 10.7 below.

```
[**] [1:615:3] SCAN SOCKS Proxy attempt [**]
[Classification: Attempted Information Leak] [Priority: 2]
01/05-20:21:39.774229 30.30.30.10:45315 -> 172.16.2.17:1080
TCP TTL:46 TOS:0x0 ID:24625 IpLen:20 DgmLen:40
******S* Seq: 0x304154CB  Ack: 0x0  Win: 0x1000  TcpLen: 20
[Xref => http://help.undernet.org/proxyscan/]

[**] [111:12:1] spp_stream4: NMAP FINGERPRINT (stateful) detection [**]
01/05-20:21:40.565593 30.30.30.10:45325 -> 172.16.2.17:135
TCP TTL:46 TOS:0x0 ID:41568 IpLen:20 DgmLen:60
***A**** Seq: 0x26DA7072  Ack: 0x0  Win: 0x1000  TcpLen: 40
TCP Options (5) => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL
```

Figure 10.7 Some of Snort's alerts for the SYN stealth with TCP + ICMP Ping Discovery option attack scenario.

From the results shown in Figure 10.6 and Figure 10.7, it is evident that this attack scenario actually scans the network and that Snort alerts on this scanning activity.

In the other cases (like all the LIST SCAN types with different discovery methods, PING SWEEP with TCP Ping discovery method) where there were no alerts produced, there were also no scanning activity. The scanning results for these attacks scenarios are shown in Figure 10.8, Figure 10.9, and Figure 10.10 below.

nmap (V. 3.00) scan initiated Sun Jan 05 22:22:46 2003 as: nmap -sL -PT -O -T 3 -oN c:\temp\friday\listtcp 172.16.2.17
Host LAN_SNIFF_8 (172.16.2.17) not scanned
Nmap run completed at Sun Jan 05 22:22:47 2003 -- 1 IP address (0 hosts up) scanned in 1 second.

Figure 10.8 Result of the LIST SCAN with TCP Ping Discovery option attack scenario.

nmap (V. 3.00) scan initiated Sun Jan 05 22:23:12 2003 as: nmap -sL -P0 -O -T 3 -oN c:\temp\friday\listdont 172.16.2.17
Host LAN_SNIFF_8 (172.16.2.17) not scanned
Nmap run completed at Sun Jan 05 22:23:13 2003 -- 1 IP address (0 hosts up) scanned in 1 second

Figure 10.9 Result of the LIST SCAN with Don't Ping Discovery option attack scenario.

nmap (V. 3.00) scan initiated Sun Jan 05 22:27:46 2003 as: nmap -sP -PT -T 3 -oN c:\temp\friday\pswtcp 172.16.2.17
Host LAN_SNIFF_8 (172.16.2.17) appears to be up.
Nmap run completed at Sun Jan 05 22:27:46 2003 -- 1 IP address (1 host up) scanned in 0 seconds

Figure 10.10 Result of the PING SWEEP with TCP Ping Discovery option attack scenario.

10.3 Conclusions

It is evident from the above results that the Snort intrusion detection system is capable of detecting/alerting on all Nmap scanning types and discovery option where scanning activity is actually carried out for a 100% detection rate. Also, Snort does not produce any false alarms for the background traffic we used.

It is also evident from the results that Nmap under the Windows operating system does not scan the network for all of Nmap scan types when the ICMP Ping discovery method is used. One explanation for this is the fact that the network under the Windows operating system does not allow ICMP echo requests (or responses) packets. This is evident in the blocking of probes when this command option is used.

Also, Nmap under List Scan does not perform scanning activity. This is evident from the results obtained. Nmap documentation explains that the List Scan simply list the IP addresses/Names on hosts on the network without actually scanning them.

About 73% of all the Nmap scan types and discovery methods investigated in this work resulted in scanning activity.

The Perl script developed in this work is easy to expand, portable and efficient.

10.4 Future Work

The next step planned for this work is fusion system. As discussed in Chapter 8, fusion systems build scenarios of alerts into groups, and determine the scenario membership of a new alert given the alerts and the groups already present, based on probability calculations or correlation techniques. The different Snort alerts collected in the process of running the many different attacks scenarios in the course of this work have been made available for this purpose.

APPENDIX A

SNORT INTRUSION DETECTION SYSTEM FALSE ALARM RATE TESTS SCHEDULE

In this Appendix, false alarm rate tests schedule is presented.

Table A.1 Snort False Alarm Rate Tests Schedule

System	Background Traffic Used	Traffic Duration	# of False Alarms
Snort	Client-Server TCP	24 hours	0
Snort	Client-Server UDP	24 hours	0
Snort	Fault Traffic	24 hours	0
Snort	FTP Traffic	24 hour	0
Snort	HTTP Traffic	24 hours	0

APPENDIX B

SNORT INTRUSION DETECTION SYSTEM DETECTION RATE USING NMAP PROGRAMS TESTS SCHEDULE

In this Appendix, detection rate tests schedule is presented.

Table B.1 Snort Detection Rate Using Nmap Programs Tests Schedule

Scan Type and Discovery Option	Attack Duration	Snort Alerts
SYN – TCP Ping	5 mins	25
SYN – TCP + ICMP	5 mins	18
SYN – ICMP Ping	5 mins	0
SYN – Don't Ping	5 mins	10
FIN – TCP Ping	5 mins	4177
FIN – TCP + ICMP	5 mins	1254
FIN – ICMP Ping	5 mins	0
FIN – Don't Ping	5 mins	1611
PING SWEEP – TCP Ping	5 mins	0
PING SWEEP – TCP + ICMP	5 mins	56
PING SWEEP – ICMP Ping	5 mins	72
PING SWEEP – Don't Ping	5 mins	0
UDP SCAN – TCP Ping	5 mins	13
UDP SCAN– TCP + ICMP	5 mins	21
UDP SCAN – ICMP Ping	5 mins	0
UDP SCAN – Don't Ping	5 mins	11
NULL SCAN – TCP Ping	5 mins	5934
NULL SCAN – TCP + ICMP	5 mins	2503
NULL SCAN – ICMP Ping	5 mins	0
NULLSCAN–Don't Ping	5 mins	1610
XMAS TREE – TCP Ping	5 mins	5913
XMAS TREE – TCP + ICMP	5 mins	2794
XMAS TREE – ICMP Ping	5 mins	0
XMAS TREE – Don't Ping	5 mins	1611
IPPROTOCOLSCAN–TCP Ping	15 mins	38
IPPROTOCOLSCAN–TCP+ICMP	15 mins	45
IP PROTOCOLSCAN – ICMP Ping	15 mins	0
IP PROTOCOL SCAN – Don't Ping	15 mins	6
ACK SCAN – TCP Ping	15 mins	47
ACKSCAN– TCP + ICMP	15 mins	340
ACK SCAN – ICMP Ping	15 mins	0
ACK SCAN – Don't Ping	15 mins	9
WINDOW SCAN – TCP Ping	15 mins	26
WINDOW SCAN – TCP + ICMP	15 mins	21
WINDOW SCAN – ICMP Ping	15 mins	0
WINDOW SCAN – Don't Ping	15 mins	9
RCP SCAN – TCP Ping	15 mins	113
RCPSCAN – TCP + ICMP	15 mins	168
RCP SCAN – ICMP Ping	15 mins	0
RCP SCAN – Don't Ping	15 mins	10
LIST SCAN – TCP Ping	15 mins	0
LIST SCAN – TCP + ICMP	15 mins	0
LIST SCAN – ICMP Ping	15 mins	0
LIST SCAN – Don't Ping	15 mins	0
CONNECT – TCP Ping	15 mins	30
CONNECT – TCP + ICMP	15 mins	45
CONNECT – ICMP Ping	15 mins	0
CONNECT – Don't Ping	15 mins	22

APPENDIX C

SNORT INTRUSION DETECTION SYSTEM DETECTION RATE USING MANUAL ATTACK PROGRAMS TESTS SCHEDULE

In this Appendix, the tests schedule for determining the detection rate of Snort using the Manual Application Programs is presented.

Table C.1 Detection Rate Tests Schedule for Snort Using Manual Attack Application Programs

Attack Name	Attack Sub-Name	Attack Other Name	Duration	Snort
dddsping			5mins	30
Battlepong			5mins	2728
Bloodlust			5mins	63
Divine	RETRiBUTION	FyRE	5mins	175
		BRiMSTONE	5mins	0
	PLAGUEz	FLOODz	5mins	59
		LOCUSTs	5mins	0
	OMENz	HELLFyRE	5mins	0
		DEMONz	5mins	0
	INCANTATIONz	HAiLSTORM	5mins	0
		WHyRLWiND	5mins	0
Elite	anarchy		5mins	430
	connections		5mins	0
	flood		5mins	0
	nuke		5mins	931
	ping		5mins	0
	pingflood		5mins	0
	teardrop		5mins	188
Iping32	iping32		5mins	485
Packetbuild	PckBuilder		5mins	0
Ping-G	PIN-G		5mins	0
Rocket10	Rocketv1_0		5mins	5546
Vai-te-ja-icmpbomber	Vai-te Ja ICMP Bomber		5mins	0
WinNuke	WinNUKE		5mins	1135

APPENDIX D

INTRUSION DETECTION SYSTEM DETECTION RATE USING THE DIFFERENT ATTACK SCENARIOS TESTS SCHEDULE

In this Appendix, the tests schedule that was followed in determining the detection rate of the Snort intrusion detection system using the attack scenarios is presented.

Week 1
Day 1 – Original attack script
Day 2 – Original attack script continues
Day 3 – Stealth attacks scenarios based on timing modes
Day 4 – Stealth attacks scenarios based on timing modes continues
Day 5 – Stealth attacks scenarios based on scan delay
Day 6 – Stealth attacks scenarios based on scan delay continues
Day 7 – Stealth attacks scenarios based on scan delay continues

APPENDIX E

INFORMATION GATHERING SCRIPTS

In this Appendix, the Perl configuration files and Perl script used in this phase of the program are presented.

(1) pre_exp.conf

In this Appendix, the Perl configuration files are presented.

```
#!/usr/bin/perl

# This is the configuration file of pre_exp.pl. It is a Perl script indeed
# so it is called by "do 'pre_exp.conf'" in pre_exp.pl. It defines
# parameters for pre_exp.pl to run to exploit which machines are up and
# what services are open. Some of the parameters may be defined in scenario.pl
# as well.

# Define exploit technique. Change it in Windows.
$PRE_EXP_CMD = "nmap";

# Define destination network range.
#$PRE_EXP_DIP = "172.16.2.1/24";
$PRE_EXP_DIP = "172.16.2.1/24"; # ?? with /27 and delay 2000, sendconnecttcpquery:
Could not scavenge a free socket! ??

# Service scan range.
$PRE_EXP_DPORT_TCP = "20-30,80,110";
$PRE_EXP_DPORT_UDP = "53,111";

# If you want to add a scan delay to make the scan more stealth.
# The unit is millisecond. Set it to 0 or comment it if you don't want delay.
$PRE_EXP_SCAN_DELAY = 0;

# If you want to spoof source IP address.
# Note in Linux you have to have the spoofed IP address bound in your network
# interface. Don't set it if IP address is not spoofed.
#$PRE_EXP_SIP = "10.1.2.3";

# If you want to set source port.
# Comment it if you don't want set it.
$PRE_EXP_SPORT = 1500;
```

(2) The Perl Script

```
#!/usr/bin/perl -w

# This file is ran first because it tries to exploit which
# machines are up and what services are open in a destination network range.
# After that, we have some understandings of the destination network so that
# further attacks will be exploited. Of course, pre_exp.pl can be run at
# any stage a scenario.
# The technique to do this is using Nmap.

#----------------------------------------------------------------------------
# read configuration file.
#----------------------------------------------------------------------------
(-r "pre_exp.conf") || die "Cannot open pre_exp.conf: $!";
BEGIN { do "pre_exp.conf"; }

#----------------------------------------------------------------------------
# check which hosts are up in the destination addresses.
# print only the IP address of the hosts which are up in a temporary file.
# using: nmap ... -n -sP -PS <target IP address range>
#                 |  |  + use SYN (not ACK scan) for root
#                 |  + ping sweep
#                 + numeric address is used only
#----------------------------------------------------------------------------
@cmd = ($PRE_EXP_CMD);
if(defined($PRE_EXP_SPORT) && $PRE_EXP_SPORT != 0) {
  push(@cmd, "-g $PRE_EXP_SPORT");
}
if(defined($PRE_EXP_SCAN_DELAY) && $PRE_EXP_SCAN_DELAY != 0) {
  push(@cmd, "--scan_delay $PRE_EXP_SCAN_DELAY");
}
push(@cmd, "-n", "-sP -PS $PRE_EXP_DIP"); # "-sP -PS $PRE_EXP_DIP" is one field.
push(@cmd, ">pre_exp.data.tmp");

# system generally return non-zero if the command went wrong.
system("@cmd") && die "$PRE_EXP_CMD running error: $?\n";

open(FP_TMP,"<pre_exp.data.tmp") or die "<1> Cannot open pre_exp.data.tmp: $!";
open(FP_OUT,">pre_exp.data") or die "<1> Cannot open pre_exp.data: $!";
while(defined($line=<FP_TMP>)) {
  chomp($line);
  if($line =~ s/^Host.*\(([\d\.]+)\).*$/$1/) {
    print FP_OUT "$line\n";
  }
}
```

```
pop(@cmd);
my ($sec,$min,$hour,$mday,$mon,$year) = (localtime)[0,1,2,3,4,5];  # use ()
#print FP_OUT "\nTest up: \"@cmd\" on ", `date +\"%a %D %T\"`;
printf FP_OUT ("\nTest up: \"@cmd\" on %02d/%02d/%02d %02d:%02d:%02d\n",
$mon+1, $mday, $year%100, $hour, $min, $sec);
close(FP_TMP);
close(FP_OUT);

#---------------------------------------------------------------------------
# for each IP address, test what possible services are running on it.
# save those services with IP address to the database.
# using: nmap ... -n -sS -P0 <target IP address> -p <listed-service-ports>
# using: nmap ... -n -sT -P0 <target IP address> -p <listed-service-ports>
#                 |  | + don't try to ping first
#                 |  + non-superuser can only use -sT.
#                 + numeric address is used only
# run above command with -sU to test UDP port.
#---------------------------------------------------------------------------

pop(@cmd); # command becomes like: nmap -g 1500 --scan_delay 1 -n
push(@cmd, "-sS -P0");

rename("pre_exp.data","pre_exp.data.tmp") || die "<1> Cannot rename: $!";
open(FP_TMP,"<pre_exp.data.tmp") or die "<2> Cannot open pre_exp.data.tmp: $!";
open(FP_OUT,">pre_exp.data") or die "<2> Cannot open pre_exp.data: $!";


while(defined($addr_tmp=<FP_TMP>)) {
  chomp($addr_tmp);
  if($addr_tmp =~ s/^([\d\.]+)$/$1/) {
    push(@cmd,"$addr_tmp");
#--------------+
# test TCP port|
#--------------+
    if( defined($PRE_EXP_DPORT_TCP) ) {
      push(@cmd, "-p $PRE_EXP_DPORT_TCP");
    }
    @result = `@cmd`; # run the scan program

    chomp(@result);
    @open_list=@filtered_list=(); # @closed_list=(); don't print the closed
    foreach $line (@result) {
      if($line =~ /^(\d+)\/tcp.*open.*$/) {
        push(@open_list,$1);
      }
```

```
            elsif($line =~ /^(\d+)\/tcp.*filtered.*$/) {
                push(@filtered_list,$1);
            }
        }
        pop(@cmd);
#--------------+
# test UDP port|
#--------------+
        if( defined($PRE_EXP_DPORT_UDP) ) {
            push(@cmd, "-p $PRE_EXP_DPORT_UDP");
        }
        push(@cmd, "-sU");
        @result = `@cmd`; # run the scan program

        chomp(@result);

        @udp_open_list=@udp_filtered_list=(); # @udp_closed_list=();
        foreach $line (@result) {
            if($line =~ /^(\d+)\/udp.*open.*$/) {
                push(@udp_open_list,$1);
            }
            elsif($line =~ /^(\d+)\/udp.*filtered.*$/) {
                push(@udp_filtered_list,$1);
            }
        }

        pop(@cmd); pop(@cmd); pop(@cmd);
#--------------+
# print result |
#--------------+
        printf FP_OUT ("%-16s",$addr_tmp);
        print FP_OUT "| @open_list : @filtered_list | @udp_open_list : @udp_filtered_list
|\n";
    }
    else { # if-else, why cannot use elsif here ?
        print FP_OUT "$addr_tmp\n";
    }
}

($sec,$min,$hour,$mday,$mon,$year) = (localtime)[0,1,2,3,4,5];   # use ()
printf FP_OUT ("Test TCP port: \"@cmd <IP> -p $PRE_EXP_DPORT_TCP\" on
%02d/%02d/%02d %02d:%02d:%02d\n", $mon+1, $mday, $year%100, $hour, $min,
$sec);
($sec,$min,$hour,$mday,$mon,$year) = (localtime)[0,1,2,3,4,5];   # use ()
```

```
printf FP_OUT ("Test UDP port: \"@cmd <IP> -p $PRE_EXP_DPORT_UDP -sU\" on
%02d/%02d/%02d %02d:%02d:%02d\n", $mon+1, $mday, $year%100, $hour, $min,
$sec);

#print FP_OUT "Test TCP port: \"@cmd <IP> -p $PRE_EXP_DPORT_TCP\" on ",
`date +\"%a %D %T\"`;
#print FP_OUT "Test UDP port: \"@cmd <IP> -p $PRE_EXP_DPORT_UDP -sU\" on ",
`date +\"%a %D %T\"`;

close(FP_TMP);
close(FP_OUT);

unlink("pre_exp.data.tmp") || warn "warning on deleting file: $!";
```

APPENDIX F

ATTACK EXPLOIT SCRIPTS

In this Appendix, the Perl script and the configuration files in this phase of the program are presented.

(1) The Perl script

```
#!/usr/bin/perl

# This is the scenario building batch program. It reads configuration
# information of each attack scenario from sce_exp.conf and run the scenario.
# Using fork() to dispatch each scenario so that they can run simultaneously.
# There is may be some waiting time between scenarios, as indicated in the
# configuration file.
#

#---------------------------------------------------------------------------
# open sce_cmd.conf and sce_exp.conf to read configuration information
#---------------------------------------------------------------------------
open (FP_CMDC,"sce_cmd.conf") or die "Cannot open configuration file: $!";
while (defined($line=<FP_CMDC>)) {
   next if ($line =~ /^#/);                    # skip comments
   next if ($line =~ /^\s*$/);                  # skip blank lines
   chomp($line);
   my @tmp_lv = split(/\s+/,$line);
   my ($tmp_key, @tmp_value) = @tmp_lv;
   $cmd_hv{$tmp_key} = join (' ', @tmp_value);        # better use []?
}
close(FP_CMDC);

open (FP_CONF,"sce_exp.conf") or die "Cannot open configuration file: $!";

LOOP: while (defined($line=<FP_CONF>)) {
   next LOOP if ($line =~ /^#/);                 # skip comments
   next LOOP if ($line =~ /^\s*$/);               # skip blank lines
   chomp($line);

   if ($line =~ /^\s*([\d\.]+)\s*([smh])\s*$/) {
      if ($2 eq 's')   { sleep int($1); }          # use eq, not the ==
      elsif ($2 eq 'm') { sleep int($1*60); }       # use eq, not the ==
      elsif ($2 eq 'h') { sleep int($1*60*60); }     # use eq, not the ==
print "waiting for $1$2\n";
   }
```

```
   elsif ($line =~ /^\s*newdata*$/) {
print "updating database.\n";
      do "pre_exp.pl";
   }
   else {                               # it's a attack command
      @cmd=""; $repetition = 1;
      process_parameter();

      shift(@cmd);       # remove the first "".
#print "cmd: @cmd\n";
#print "repetition: $repetition\n";

#---------------------------------------------------------------------------
# fork...
#---------------------------------------------------------------------------
      if(!defined($child_pid=fork())) {
         die "Cannot fork another process to run the scenarios: $!";
      }
      elsif ($child_pid == 0) {         # this is child process
print "command: @cmd\n";
#system("@cmd");
         $tmp_repetition = $repetition;
         while (--$tmp_repetition > 0) {
            @cmd="";
            process_parameter();
            shift(@cmd); # remove the first "".
print "command: @cmd\n";
#system("@cmd");
         }

         exit;
      }
      else {                           # this is parent process.
#wait;
      }
   }
}

close(FP_CONF);

#-----------------------------------------------------+
# construct a hash from the database: key IP, value port |
#-----------------------------------------------------+
sub read_database {
   my ($line, @tmp_lv);            # need parentheses.
   open (FP_PRE,"pre_exp.data") or die "Cannot open pre-scann data file: $!";
```

```
   while(defined($line=<FP_PRE>)) {
#      @tmp_lv="";     # "" or even undef is an element, so use pop later.
      chomp($line);
      last unless ($line =~ /^\d/);  # break if not beginning with a digit
      $line =~ /^([\d\.]+)\s+\| (.*) : (.*) \| (.*) : (.*) \|.*$/;
      push(@tmp_lv,$2,$3,$4,$5);
      $tmp_hv{$1} = [@tmp_lv];       # must use []. %tmp_hv is global then.
      pop(@tmp_lv); pop(@tmp_lv); pop(@tmp_lv); pop(@tmp_lv);
   }
   close (FP_PRE);
}

sub process_parameter {
   my @tmp_line = split(/\s+/, $line);
   my @cmd_format = split(/\s+/, $cmd_hv{$tmp_line[0]});
   unless (@cmd_format) {
      warn "Wrong entry word: $tmp_line[0]. Ignored.";
      next LOOP;
   }
   push (@cmd, shift(@cmd_format));             # command name

   shift(@tmp_line);
   $d_ip_or_pt = 0;
   foreach my $parameter (@tmp_line) {
      if ($parameter =~ /^D/) {
         if ($d_ip_or_pt == 0) {              # parameter is ip
            unless (defined %tmp_hv) { read_database(); }
            $tgt_para = $tgt_ip = ip_from_database($parameter);
            $d_ip_or_pt = 1;
         }
         else {                           # parameter is ip
            $tgt_para = $tgt_pt = pt_from_database($parameter);
         }
      }
      elsif ($parameter =~ /^R/) {
         $_ = $parameter;
         my $func = /^(R+)/    ? $1 : undef;
         my $low  = /\((\d+)-/ ? $1 : undef;
         my $high = /-(\d+)\)/ ? $1 : undef;

         $tgt_para =
            $func =~ /^R$/  ? (defined $low && defined $high) ? func_r($low, $high)  :
func_r()  :
            $func =~ /^RR$/ ? (defined $low && defined $high) ? func_rr($low, $high) :
func_rr() : warn "wrong R field: $!";
```

```
        }
        else {
           $tgt_para = $parameter;
        }

        if ($tgt_para =~ /^\[\[(.*)\]\]$/) {          # [[other parameter field]]
           push(@cmd, join(" ", split(/\.\./,$1)));
        }
        elsif ($tgt_para =~ /^{{(.*)}}$/) {          # {{repetition field}}
           $repetition = $1 =~ /^(\d+)$/     ? $1            : # must use ()
                        $1 =~ /^(\d+)-(\d+)/ ? func_r($1, $2) : $repetition;
        }
        else {                    # filed corresponding to command conf
           $_ = shift(@cmd_format);
           if ($_) {
              if (/^-$/) { push(@cmd,$tgt_para); }
              else {
                 if (/NS$/) {
                    s/NS$//;
                    push(@cmd, $_.$tgt_para);
                 }
                 else { push(@cmd, $_." ".$tgt_para); }
              }
           }
           else { warn "nonmatched field: $!"; }
        }
     }

}

#-------------------------------------------------------------------------
# set target IP address (range) and port number(s) from pre_exp.data.
#-------------------------------------------------------------------------

sub ip_from_database {
   $_ = $_[0];
   return ( /^DR$/  ? func_dr(keys %tmp_hv)  :       # ip DR
            /^DRR$/ ? func_drr(keys %tmp_hv) :       # ip DRR
            /^DA$/  ? func_da(keys %tmp_hv)  : $_ );  # ip DA
}

sub pt_from_database {
   my $tmp_ip = (split(/\s+/,$tgt_ip))[0]; # get the 1st ip no matter DR/DRR/DA
   $_ = $_[0];

   my $tmp_pt =
```

```
    /^DR$/  ? $tmp_hv{$tmp_ip}[0] ? func_dr(split(/\s+/,$tmp_hv{$tmp_ip}[0]))  : 80 :
    /^DRR$/ ? $tmp_hv{$tmp_ip}[0] ? func_drr(split(/\s+/,$tmp_hv{$tmp_ip}[0])) :
80 :
    /^DA$/  ? $tmp_hv{$tmp_ip}[0] ? func_da($tmp_hv{$tmp_ip}[0])           : 80 :
80;
  $tmp_pt =~ tr/ /,/;
  return $tmp_pt;
}

#---------------------------------------------------------------------------
# functions.
#---------------------------------------------------------------------------
sub func_r {
  my ($low,$high);
  $low  = defined $_[0] ? $_[0] : 20;
  $high = defined $_[1] ? $_[1] : 29;
  return int(rand($high-$low+1)) + $low;
}

sub func_rr {
  my ($low,$high);
  $low  = defined $_[0] ? $_[0] : 20;
  $high = defined $_[1] ? $_[1] : 99;
  my $temp1 = int(rand($high-$low+1)) + $low;
  my $temp2 = int(rand($high-$low+1)) + $low;
  return ($temp1 < $temp2 ? "$temp1-$temp2" : $temp1 > $temp2 ? "$temp2-$temp1" :
$temp1);
}

sub func_dr {
  my @temp = @_;
  return $temp[int(rand(@temp))];
}

sub func_drr {
  my $ratio = 0.5;                # default, 50% will be selected.
  my $temp1 = "";
  foreach my $temp2 (@_) {
    if((rand) < $ratio) { $temp1 .= ($temp2. " "); }
  }
#   ($temp1) ? chop($temp1) : $temp1=$_[$#_]; # why doesn't it work?
  if($temp1) { chop($temp1); }          # remove the last ,
  else { $temp1=$_[$#_]; }              # at least should get one.
  return $temp1;
}
```

```
sub func_da {
   return "@_";                          # any other good method ?
}
```

(2) sce_cmd.conf

```
# This file specifies each attack program and its parameter format. The first
# column is the entry name. The second is the attack program name and the
# rest columns are its parameters. The sce_exp.pl script reads an entry from
# the sce_exp.conf and will query this file to construct the command by
# finding the entries having same command name in two files.

# nmap port scan format:
# entry   cmd   ip  port  delay        source_port  type
nmap_scan  nmap  -   -p   --scan_delay  -g          -sNS

#1
syntcpn nmap - -sNS
#2
syntcpicmpn nmap - -sNS
#3
synicmpn nmap - -sNS
#4
syndontn nmap - -sNS
#5
syntcpi nmap - -sNS
#6
syntcpicmpi nmap - -sNS
#7
synicmpi nmap - -sNS
```

(3) sce_exp.conf

```
# This is the configuration file of scenario running program, sce_exp.pl.
# Each line stands for an attack with parameters except for the line beginning
# with # (it is a comment).

# There are three types of statement: attacks, waiting time between attack scenarios and
#running pre-scanning program to generate new pre_exp.data.
# The format of attacks is:                    program_name <parameters>
# The format of time waiting is:                 <n>s, <n>m, or <n>h
# The format of updating new database file is:    newdata

# Example: format of port scan using nmap.
```

```
# program target_addr     services          delay    source_port   type
# nmap    DR/DRR/DA/value  R/RR/DR/DRR/value  R/value  R/value       S/T/X/N
#nmap_scan DA R(3-5) 2000 1500 S [[-n..-P0]] {{6-9}}
#nmap_ping_sweep DR R(1500-1600) P [[-n..-PS]] {{4}}
#nmap_scan DRR R(1-5) 2000 1500 S [["aa"..bb]] {{2}}
#3.2s
#nmap_scan DR DR R(1000-2000) 1500 S {{4}}
#1.2m
#newdata
# I added the stuff below
#1.0m
#netcat_port DR DR
2s
#1
syntcpn 172.16.2.11 S [[-PT..-O..-T..0]] {{1}}
50s
#2
syntcpicmpn 172.16.2.12 S [[-PT..-PI..-O..-T..0]] {{2}}
55s
#3
synicmpn 172.16.2.13 S [[-PI..-O..-T..0]] {{5}}
100s
#4
syndontn 172.16.2.17 S [[-P0..-O..-T..0]] {{7}}
80s
#5
syntcpi 172.16.2.12 S [[-PT..-O..-T..0]] {{5}}
51s
#6
syntcpicmpi 172.16.2.13 S [[-PT..-PI..-O..-T..0]] {{2}}
55s
#7
synicmpi 172.16.2.17 S [[-PI..-O..-T..0]] {{5}}
```

APPENDIX G

BACKGROUND TRAFFIC SCRIPTS

In this Appendix, the background traffic scripts are presented.

(1) bkgrd_cmd.conf

```
# This is the configuration file of background traffic types. There are currently four types
# of background traffic. They are FTP, Telnet, Http and Mail services and these cover
# most of the common traffic in practice.

[FTP_1]
command = ftp
host    = 172.16.2.13
account = anonymous
passwd  = come
[[local files]]
c:\cygwin\home\administrator\p13.jpg
c:\cygwin\home\administrator\p130.jpg
c:\cygwin\home\administrator\p131.jpg
c:\cygwin\home\administrator\p132.jpg
[[remote files]]
c:\inetpub\ftproot\p1.jpg
c:\inetpub\ftproot\p10.jpg
c:\inetpub\ftproot\p100.jpg
c:\inetpub\ftproot\p101.jpg
c:\inetpub\ftproot\p102.jpg
c:\inetpub\ftproot\p103.jpg
c:\inetpub\ftproot\p104.jpg
c:\inetpub\ftproot\p105.jpg
c:\inetpub\ftproot\p106.jpg
c:\inetpub\ftproot\p107.jpg
####################
[Telnet_1]
command = telnet
host    = 172.16.2.18
account = sniffer
passwd  = sniffer
delay   = 2
[[commands]]
ls
cd:ls -l
####################
#[Http_1]
#command = lynx
```

```
#[[sites]]
#www.njit.edu
#www.yahoo.com
###################
#[Mail_1]
#command = mail
#[[address]]
#a@b.com
#c@d.net
#[[messages]]
#msg1
#msg2:msg3

###################

[End]
```

(2) bkgrd_gen

```
# This is the configuration file of bkgrd_gend.pl, the script to generate
# background traffic. There are two types of statement in this file, one
# is the background traffic entry and the other is time between two
# background traffic entries.

# waiting line: <n><s/m/h>

# ftp_entry  local(-/R/RR/All/value)  remote(-/R/RR/All/value)  repetition
FTP_1      -                  R                    {{1}}
5s
# telnet_entry  command_list  repetition
Telnet_1      A          {{1}}

# http_entry   site_list  repetition
#Http_1        R        {{2}}
#Http_1        A        {{2}}

# mail_entry    address_list message_list  repetition
#Mail_1        R          R            {{1}}
```

(3) The script

```
# This script generates background traffic according to its configuration file,
# i.e., bkgrd_gen.conf and bkgrd_cmd.conf.


$cmd_conf_fn = "bkgrd_cmd.conf";
$conf_fn     = "bkgrd_gen.conf";


open (FP_CMDC, $cmd_conf_fn) or die "Cannot open cmd configuration file :$!";
while (defined($line=<FP_CMDC>)) {
   next if ($line =~ /^#/);                 # skip comments
   next if ($line =~ /^\s*$/);               # skip blank lines
   chomp($line);

   if ($line =~ /^\[([^\[\]]+)\]$/) {
      $key = $1; $field = 0;

      if(defined $key_pre) {
         $cmd_hv{$key_pre} =
            $key_pre =~ /^FTP/i    ? [$command, $host, $account, $passwd, [@ftp_local],
[@ftp_remote]] :
            $key_pre =~ /^Telnet/i ? [$command, $host, $account, $passwd, $delay,
[@telnet_cmd]]    :
            $key_pre =~ /^Http/i   ? [$command, [@http_site]]                          :
            $key_pre =~ /^Mail/i   ? [$command, [@mail_addr],
[@mail_msg]]                       : undef;
      }
      @ftp_local = @ftp_remote = @telnet_cmd = @http_site = @mail_addr =
@mail_msg = "";
      shift(@ftp_local); shift(@ftp_remote); shift(@telnet_cmd); shift(@http_site);
shift(@mail_addr); shift(@mail_msg);

      $key_pre = $key;
      next;
   }

   if ($key =~ /^FTP/i) {
      $_ = $line;
      if   (/^command\s*=\s*([^\s]+)/) { $command = $1; }
      elsif (/^host\s*=\s*([^\s]+)/)    { $host    = $1; }
      elsif (/^account\s*=\s*([^\s]+)/) { $account = $1; }
      elsif (/^passwd\s*=\s*([^\s]+)/)  { $passwd  = $1; }
      elsif (/^\[\[.*\]\]/)             { $field++;     } # comm/local/remote
      else { $field == 1 ? push(@ftp_local, $line) : push(@ftp_remote, $line); }
```

```
  }
  elsif ($key =~ /^Telnet/i) {
    $_ = $line;
    if   (/^command\s*=\s*([^\s]+)/) { $command = $1; }
    elsif (/^host\s*=\s*([^\s]+)/)    { $host    = $1; }
    elsif (/^account\s*=\s*([^\s]+)/) { $account = $1; }
    elsif (/^passwd\s*=\s*([^\s]+)/)  { $passwd  = $1; }
    elsif (/^delay\s*=\s*([^\s]+)/)   { $delay   = $1; }
    elsif (/^\[\[.*\]\]/)             { $field++;     } # comm/cmd
    else { $field == 1 and push(@telnet_cmd, $line) }
  }
  elsif ($key =~ /^Http/i) {
    $_ = $line;
    if   (/^command\s*=\s*([^\s]+)/) { $command = $1; }
    elsif (/^\[\[.*\]\]/)             { $field++;     } # comm/sites
    else { $field == 1 and push(@http_site, $line) }
  }
  elsif ($key =~ /^Mail/i) {
    $_ = $line;
    if   (/^command\s*=\s*([^\s]+)/) { $command = $1; }
    elsif (/^\[\[.*\]\]/)             { $field++;     } # comm/message
    else { $field == 1 ? push(@mail_addr, $line) : push(@mail_msg, $line); }
  }
#   elsif ($key =~ /^End/i) { last; }
}
$key = $field = undef;
$command = $host = $account = $passwd = $delay = undef;
@ftp_local = @ftp_remote = @telnet_cmd = @http_site = @mail_addr = @mail_msg =
undef;
close(FP_CMDC);

open (FP_CONF,$conf_fn) or die "Cannot open configuration file: $!";

while (defined($line=<FP_CONF>)) {
  next if ($line =~ /^#/);                # skip comments
  next if ($line =~ /^\s*$/);             # skip blank lines
  chomp($line);

  if ($line =~ /^\s*([\d\.]+)\s*([smh])\s*$/) {
print "waiting for $1$2\n";
    if ($2 eq 's')   { sleep int($1); }        # use eq, not the ==
    elsif ($2 eq 'm') { sleep int($1*60); }    # use eq, not the ==
    elsif ($2 eq 'h') { sleep int($1*60*60); } # use eq, not the ==
  }
  else {
    @ftp_list = @telnet_list = @http_list = @mail_list = "";
```

```
    shift(@ftp_list); shift(@telnet_list); shift(@http_list); shift(@mail_list);
    $_ = $line;
    if (/^FTP/i) {
      ($entry, my ($ftp_local, $ftp_remote), $repetition) = split;
      $repetition = defined $repetition ? process_repetition($repetition) : 1;

      for (my $i=0; $i<$repetition; $i++) {
        $_ = $ftp_local; my @ftp_local = ""; shift @ftp_local;
        unless (/^-$/) {    # must use () here.
          push (@ftp_local, /^R$/  ? func_r($cmd_hv{$entry}[4])  :
                            /^RR$/ ? func_rr($cmd_hv{$entry}[4]) :
                            /^A$/  ? func_a($cmd_hv{$entry}[4])  : $_);
        }

        $_ = $ftp_remote; my @ftp_remote = ""; shift @ftp_remote;
        unless (/^-$/) {    # must use () here.
          push (@ftp_remote, /^R$/  ? func_r($cmd_hv{$entry}[5])  :
                             /^RR$/ ? func_rr($cmd_hv{$entry}[5]) :
                             /^A$/  ? func_a($cmd_hv{$entry}[5])  : $_);
        }
        push (@ftp_list, [[@ftp_local], [@ftp_remote]]);
#print "+++ @ftp_local, @ftp_remote +++\n";
# then $ftp_list[$repetition][local/remote][index]. $tmp = $ftp_list[0][0]; scalar(@$tmp)
is the size.
      }
    }
    elsif (/^Telnet/i) {
      ($entry, my $telnet_cmd, $repetition) = split;
      $repetition = defined $repetition ? process_repetition($repetition) : 1;

      for (my $i=0; $i<$repetition; $i++) {
        $_ = $telnet_cmd; my @telnet_cmd = ""; shift @telnet_cmd;
        unless (/^-$/) {
          push (@telnet_cmd, /^R$/  ? func_r($cmd_hv{$entry}[5])  :
                             /^RR$/ ? func_rr($cmd_hv{$entry}[5]) :
                             /^A$/  ? func_a($cmd_hv{$entry}[5])  : $_);
        }
        push (@telnet_list, [[@telnet_cmd]]);
#print "+++ $telnet_list[$i][0][0] +++\n";
      }
    }
    elsif (/^Http/i) {
      ($entry, my $http_site, $repetition) = split;
      $repetition = defined $repetition ? process_repetition($repetition) : 1;

      for (my $i=0; $i<$repetition; $i++) {
```

```
            $_ = $http_site; my @http_site = ""; shift(@http_site);
            unless (/^-$/) {
               push (@http_site, /^R$/  ? func_r($cmd_hv{$entry}[1])  :
                                 /^RR$/ ? func_rr($cmd_hv{$entry}[1]) :
                                 /^A$/  ? func_a($cmd_hv{$entry}[1])  : $_);
            }
            push (@http_list, [[@http_site]]);
#print "+++ $http_list[$i][0][0] +++\n";
         }
      }
      elsif (/^Mail/i) {
         ($entry, my ($mail_addr, $mail_msg), $repetition) = split;
         $repetition = defined $repetition ? process_repetition($repetition) : 1;

         for (my $i=0; $i<$repetition; $i++) {
            $_ = $mail_addr; my @mail_addr = ""; shift(@mail_addr);
            unless (/^-$/) {
               push (@mail_addr, /^R$/  ? func_r($cmd_hv{$entry}[1])  :
                                 /^RR$/ ? func_rr($cmd_hv{$entry}[1]) :
                                 /^A$/  ? func_a($cmd_hv{$entry}[1])  : $_);
            }

            $_ = $mail_msg; my @mail_msg = ""; shift(@mail_msg);
            unless (/^-$/) {
               push (@mail_msg,  /^R$/  ? func_r($cmd_hv{$entry}[2])  :
                                 /^RR$/ ? func_rr($cmd_hv{$entry}[2]) :
                                 /^A$/  ? func_a($cmd_hv{$entry}[2])  : $_);
            }
            push (@mail_list, [[@mail_addr], [@mail_msg]]);
#print "+++ $mail_list[$i][0][0] +++\n";
         }
      }
      else { warn "wrong or empty field: $!" }

   }

#----------+
# fork...  |
#----------+

   if(!defined($child_pid=fork())) {
      die "Cannot fork another process to run the scenarios: $!";
   }
   elsif ($child_pid == 0) {             # this is child process
      $entry =~ /^FTP/i    ? process_ftp($entry, $repetition)    :
      $entry =~ /^Telnet/i ? process_telnet($entry, $repetition) :
```

```
    $entry =~ /^Http/i   ? process_http($entry, $repetition)   :
    $entry =~ /^Mail/i   ? process_mail($entry, $repetition)   : warn "wrong or empty
field: $!";

    exit;
  }
  else {                          # this is parent process.
#wait;
  }

}

close(FP_CONF);

sub process_ftp {
  my ($entry, $repetition) = @_;
  my $tmp_fn = "ftp.tmp.$$";     # $$ is the process id.

  for my $i (0..$repetition-1) {  # same as for (..;..;) { } ?
    open (FP_TMP, ">$tmp_fn") or die "Cannot open $tmp_fn: $!";
    print FP_TMP "$cmd_hv{$entry}[0] -ni $cmd_hv{$entry}[1] <<EOF\n";
    print FP_TMP "user $cmd_hv{$entry}[2] $cmd_hv{$entry}[3]\n";
    print FP_TMP "binary\n";
    my $temp = $ftp_list[$i][0];       # local file list
    if (my $temp1 = "@$temp") {        # better way to convert list to scalar?
      $temp1 =~ tr/:/ /;
      print FP_TMP "mput $temp1\n";
    }

    $temp = $ftp_list[$i][1];          # remote file list
    if (my $temp1 = "@$temp") {        # better way to convert list to scalar?
      $temp1 =~ tr/:/ /;
      print FP_TMP "mget $temp1\n";
    }
    print FP_TMP "quit\nEOF\n";
    close (FP_TMP);

    chmod 0744, $tmp_fn;
    system ("cat $tmp_fn");
#     system ("./$tmp_fn");
  }
  unlink $tmp_fn;
}

sub process_telnet {
  my ($entry, $repetition) = @_;
```

```
    my $tmp_fn = "telnet.tmp.$$";

    for my $i (0..$repetition-1) {
        open (FP_TMP, ">$tmp_fn") or die "Cannot open $tmp_fn: $!";
        print FP_TMP "(echo $cmd_hv{$entry}[2]\nsleep $cmd_hv{$entry}[4]\n";
        print FP_TMP "echo $cmd_hv{$entry}[3]\nsleep $cmd_hv{$entry}[4]\n";

        my $comm = $telnet_list[$i][0];     # telnet command list

        my @comm_clear;                     # extended from ":"
        for $_ (@$comm) { push (@comm_clear, split /:/); }

        for my $temp (@comm_clear) {
            print FP_TMP "echo $temp\nsleep $cmd_hv{$entry}[4]\n";
        }

        print FP_TMP "exit) | $cmd_hv{$entry}[0] $cmd_hv{$entry}[1]\n";
        close (FP_TMP);

        chmod 0744, $tmp_fn;
        system ("cat ./$tmp_fn");
#       system ("./$tmp_fn");               # or system ("./$tmp_fn > tmpfile");
    }
    unlink $tmp_fn;
}

sub process_http {
    my ($entry, $repetition) = @_;

    for my $i (0..$repetition-1) {
        my $site = $http_list[$i][0];       # http site list

        my @site_clear;                     # extended from ":"
        for $_ (@$site) { push (@site_clear, split /:/); }

        for my $temp (@site_clear) {
            my @cmd = ($cmd_hv{$entry}[0], "-source", $temp);
            print "@cmd\n";
#           system "@cmd";                  # or system ("@cmd > tmpfile");
        }
    }
}

sub process_mail {
    my ($entry, $repetition) = @_;
```

```
  for my $i (0..$repetition-1) {
    my $addr = $mail_list[$i][0];      # mail address list
    my $msg  = $mail_list[$i][1];      # mail message list

    my (@addr_clear, @msg_clear);      # extended from ":"
    for $_ (@$addr) { push (@addr_clear, split /:/); }
    for $_ (@$msg)  { push (@msg_clear, split /:/);  }

    for my $temp1 (@addr_clear) {
      for my $temp2 (@msg_clear) {
        print "$cmd_hv{$entry}[0] $temp1 < $temp2\n";
#         system "$cmd_hv{$entry}[0] $temp1 < $temp2";
      }
    }
  }
}

#--------------------------------------------------------------------------
# functions.
#--------------------------------------------------------------------------

sub process_repetition {
  return ($_[0] =~ /^{{(.*)}}$/) ? $1 =~ /^(\d+)$/      ? $1                    :
                                $1 =~ /^(\d+)-(\d+)/ ? int(rand($2-$1+1)) + $1 : 1
                               : 1;
}

sub func_r {
  my $temp = $_[0];   # note this is the reference of an array. @$temp is like a list.
  return $temp->[int(rand(@$temp))]; # scalar(@$temp) or (@$temp+0) is the size.
}
sub func_rr {
  my ($temp, $ratio) = ($_[0], 0.5);           # select 50% range
  my @temp2 = ""; shift @temp2;

  foreach my $temp1 (@$temp) {
    if (rand() < $ratio) { push (@temp2, $temp1) }
  }
  unless (@temp2) { push (@temp2, $temp->[0]) }   # at least should get one. use the
first one.
  return @temp2;
}
sub func_a {
  my $temp = $_[0];
  return @$temp;
}
```

APPENDIX H

In this Appendix, a part of the stealth attacks configuration file is given.

Sce_exp.conf

```
# This is the configuration file of scenario running program, sce_exp.pl.
# Each line stands for an attack with parameters except for the line beginning
# with # (it is a comment).
#1
syntcpn 172.16.2.11 S [[-PT..-O..-T..0]] {{1}}
50s
#2
syntcpicmpn 172.16.2.12 S [[-PT..-PI..-O..-T..0]] {{2}}
55s
#3
synicmpn 172.16.2.13 S [[-PI..-O..-T..0]] {{5}}
100s
#4
syndontn 172.16.2.17 S [[-P0..-O..-T..0]] {{7}}
80s
#5
syntcpi 172.16.2.12 S [[-PT..-O..-T..0]] {{5}}
51s
#6
syntcpicmpi 172.16.2.13 S [[-PT..-PI..-O..-T..0]] {{2}}
55s
#7
synicmpi 172.16.2.17 S [[-PI..-O..-T..0]] {{5}}
100s
#8
syndonti 172.16.2.18 S [[-P0..-O..-T..0]] {{7}}
130s
#9
syntcpid 172.16.2.13 S [[-PT..-n..-T..0]] {{3}}
71s
#10
syntcpicmpid 172.16.2.17 S [[-PT..-Pn..-O..-T..0]] {{2}}
65s
#11
synicmpid 172.16.2.18 S [[-PI..-n..-T..0]] {{5}}
110s
#12
syndontid 172.16.2.11 S [[-P0..-n..-T..0]] {{7}}
etc.
```

REFERENCES

1. R. P. Lippman, D. J. Fried, I. Graf, J. W. Haines, K. R. Kendall, D. McClung, D. Weber, S. E. Webster, D. Wyschogrod, R. K. Cunningham, and M. 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, Vol. 2, 2000.

2. K. Kendall, *"A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems"*, Master's Thesis, Massachusetts Institute of Technology, 1998.

3. N. Desai, *"Optimizing NIDS Performance,"* SecurityFocus HOME Infocus: Optimizing NIDS Performance, http://online.securityfocus.com/infocus/1589, June 6, 2002.

4. A. Lizard, *"Using SNORT for Intrusion Detection,"* SNORT Review, http://web.njit.edu/~manikopo/SNORT/snort_review1.html, Wednesday, January 23, 2002.

5. G. Saoutine et al, *"Barbarians at the Gate,"* Microsoft Certified Professional Magazine Online | Barbarians at the Gate, http://www.mcpmag.com/Features/print.asp?EditorialsID=294 .

6. O. M. Dain and R. K. Cunningham, *"Building Scenarios from a Heterogenous Alert System,"* in Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 5-6 June 2001.

7. Snort Website: http://www.snort.org.

8. ZoneAlarm Website: http://www.zonelabs.com/store/content/home.jsp.

9. Netcat : http://www.atstake.com/research/tools.

10. Nmap: http://download.insecure.org/nmap.

11. WinDump: http://windump.polito.it/.

12. Ethereal Website: http://www.ethereal.com/.

13. W. R. Stevens, *TCP/IP Illustrated Volume 1: The Protocols*, Addison-Wesley, January 1999.

14. W. Stallings, *Network Security Essential: Applications and Standards*, Prentice Hall, Upper Saddle River, New Jersey, 1999.

15. S. Northcutt, *Network Intrusion Detection: An Analyst's Handbook*, New Riders, Indiana, 1999.

16. S. Nothcutt, M. Cooper, M. Fearnow and K. Frederick, *Intrusion Signatures and Analysis*, New Riders, Indiana, 2001.

17. K. M. Walker and L. C. Cavanaugh, *Computer Security Policies and SunScreen Firewalls*, Sun Microsystems Press, California, 1998.

18. A. D. Rubin, D. Geer and M. J. Ranum, *Web Security*, Wiley Computer Publishing, USA, 1997.

19. *"Computer Attacks: What They Are And How To Defend Against Them,"* http://csrc.nist.gov/SBC/PDF/NIST_ITL_Bulletin_05-99_Comp_Attacks.pdf.

20. Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, *"HIDE: A Hierarchical Network Intrusion Detection System Using Statistical Preprocessing and Neural Network Classification,"* http://web.njit.edu/~manikopo/papers.

21. Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, *"HIDE: An Anomaly Network Intrusion Detection System Utilizing Wavelet Compression,"* http://web.njit.edu/~manikopo/papers.

22. Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, *"Neural Networks in Statistical Anomaly Intrusion Detection,"* http://web.njit.edu/~manikopo/papers.

23. Z. Zhang, J. Li, C. N. Manikopoulos, J. Jorgenson, and J. Ucles, *"A Hierarchical Anomaly Network Intrusion Detection System Using Neural Network Classification,"* http://web.njit.edu/~manikopo/papers.

24. E. Skoudis, *Counter Hack: A Step-by-Step Guide to Computer Attacks and Effective Defenses*, Prentice Hall PTR, 2002.

25. P. Mell, Computer Attacks: What They Are and How to Defend Against Them, NIST, Computer Security Division, http://www.itl.nist.gov/div893/staff/mell/pmhome.html.

26. *eTrust Intrusion Detection*, http://www3.ca.com/Solutions/Product.asp?ID=163.

27. The Recorder Program Website: http://www.kratronic.com/recorder, version 4.3.

28. O. M. Dain and R. K. Cunningham, *"Building Scenarios from a Heterogenous Alert Stream,"* Proceedings of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point, NY, 5-6 June 2001.

29. P. Ning, Y.Cui and D.S. Reeves, *"Constructing Attack Scenarios through Correlation of Intrusion Alerts,"* ACM, USA, November 18-22, 2002.