# GIAC CERTIFIED INTRUSION ANALYST (GCIA)

# PRACTICAL ASSIGNMENT

**Version 2.9**

**Robert Nine**

# GIAC Practical  Assignment 1

## *Introduction*

The data for assignment one was gathered from my home network, using a dedicated sensor running TCPDump v.3.6 (PCAP v.0.6) on an Intel-based PC (Redhat v.7.1). Besides running the Redhat provided firewall software, the sensor was hardened using the Linux-Bastille (v.1.2) script to lock down unnecessary services and ports. The only port that is open on the sensor is port 22 (SSH), which is used to transfer collected data to other systems for processing.

The data was collected using TCPDump with the following settings (host my.net.215.62 –i eth0 –w <output file>) and written to a file. Every hour TCPDump was stopped and restarted using a different output file. Later the output files were read using TCPDump with the following parameters (–vv  -r <input file>).

The format of the detect data will depend on the protocol detected, all protocols for this assignment fall into one of two categories (tcp and udp). An explaination of each format is provided in the TCPDump man page:

> The general format of a **tcp protocol** line is:
> src > dst: flags data-seqno ack window urgent options
> *Src* and *dst* are the source and destination IP addresses and ports.
> *Flags* are some combination of S (SYN), F (FIN), P (PUSH) or R (RST) or a single `.'(no flags).
> *Data-seqno* describes the portion of sequence space covered by the data in this packet.
> *Ack* is sequence number of the  next data expected the other direction on this connection.
> *Window* is the number of bytes of receive buffer space available the other direction on this  connection.
> *Urg* indicates there is `urgent' data in the packet.
> *Options* are tcp options enclosed in angle brackets (e.g., <mss 1024>).
>
> Src, dst and flags are always present. The other fields depend on the contents of the packet's tcp protocol header and are output only if appropriate.
>
> **UDP format** is illustrated by this rwho packet:
> actinide.who > broadcast.who: udp 84
> This says that port who on host actinide sent a udp datagram to port who on host broadcast, the Internet broadcast address. The packet contained 84 bytes of user data.

Some UDP services are recognized (from the source or destination port number) and the higher level protocol information printed. In particular, Domain Name service requests (RFC-1034/1035) and Sun RPC calls(RFC-1050) to NFS.

I have highlighted the time stamp of each line of the collected data as a aid to the reader in determining where each entry begins.

## Detect # 1 - Port 31337 Scan

**09:59:57.986263** 62-36-155-92.dialup.uni2.es.31337 > my.net.215.62.31337: [udp sum ok] udp 18 (ttl 113, id 30485, len 46)
**09:59:58.116263** my.net.215.62 > 62-36-155-92.dialup.uni2.es: icmp: my.net.215.62 udp port 31337 unreachable for 62-36-155-92.dialup.uni2.es > my.net.215.62: [|udp] (ttl 113, id 30485, len 46) (DF) [tos 0xc0] (ttl 255, id 3328, len 74)

### 1. Source of Trace:

The data was gathered from my home network using the sensor described in the introduction.

### 2. Detect was generated by:

Me visually scanning tcpdump output (tcpdump was limited to collecting activity directly targeting the sensor, so the amount of data was not substantial).

### 3. Probability the source address was spoofed:

It is unlikely that the source IP was spoofed, scans are of limited use unless data can be sent back to the person scanning.

### 4. Description of attack:

According to the arachNIDS database there are four intrusion events that involves port 31337/udp:

- IDS397/trojan-BackOrifice1-scan [*UDP any -> 31337*]
- IDS399/trojan-active-BackOrifice1-info [*UDP any -> 31337*]
- IDS398/trojan-active-BackOrifice1-dir [*UDP any -> 31337*]
- IDS188/trojan-probe-back-orifice [*UDP any -> 31337*]

(in other vulnerabilities databases it is referred to as: CAN-1999-0660 and 2001506 ). Due to the use of the default snaplen with tcpdump, it is not possible to narrow the event down to just one of the above choices, fortunately it's not necessary. All four are often used for the same purpose, to determine whether or not the target system is running the BackOrifice trojan.

### 5. Attack mechanism:

There are a number of ways the BackOrifice trojan can be installed on a Microsoft Windows system, without the system's owners knowledge (e.g., email attachments, freeware download from the internet, after the system has been compromised via another vulnerability, etc.). Once installed, BackOrifice can give unlimited access to the machine. When BackOrifice is installed via mass mailings or freeware, the attacker does not know when a machine has been compromised (BackOrifice does not contact the attacker like some trojans do). In order to use the BackOrifice trojan, attackers must scan for Microsoft Windows machines that are listening on port 31337. Once found, and assuming no one else has located the box first and changed the default password, the scanner now has unlimited access to the machine.

**6. Correlation:**

The information provided at www.whitehats.com is confirmed by the developers of BackOrifice, the Cult of the Dead Cow at www.cultdeadcow.com/tools/bo.html

**7. Evidence of active targeting:**

This type of reconnaissance is seldom run against a single machine. I have no reason to suspect that I was singled out.

**8. Severity:**

The severity of the attack is a –3.

(Criticality + Lethality) – (System Countermeasures + Network Countermeasures) = Severity

Criticality = 3. Had this dedicated sensor been part of an enterprise IDS system, a compromise would effectively blind us to other activity against our systems.
Lethality = 1. Had this been a Microsoft Windows machine the lethality would have been much higher, however, as the system was running Linux there is no threat from BackOrifice.
System Countermeasures = 5. With port 31337 being blocked by a software firewall on the sensor, and the fact that the exploit targets a totally different operating system, there is no chance of success.
Network Countermeasures = 2. Because of the number of exploits involving port 31337, this type of activity is easily spotted by most signature-based IDS (and in my case, easily noted in the tcpdump output).

**9. Defensive recommendation:**

If you are running any of the Microsoft operating systems, you need to block port 31337 at the network firewall, scan your system regularly using any of the standard anti-virus software packages (they all detect BackOrifice), and install a personal firewall that detect outbound as well as inbound Internet connections (such as ZoneAlarm) on each system.

**10. Multiple choice test question:**

Though there are many exploits that listen on port 31337, which is the most common:

a) t0rn root kit
b) SADMIND worm
c) BackOrifice

d) SubSeven v.1

The answer is c.

## Detect # 2 - Port 111 Scan

```
08:54:33.016263 200.204.153.224.4136 > my.net.215.62.sunrpc: S 1846790862:1846790862(0) win 32120
<mss 1460,sackOK,timestamp 15343585[|tcp]> (DF) (ttl 49, id 31435, len 60)
08:54:33.146263 my.net.215.62 > 200.204.153.224: icmp: my.net.215.62 tcp port sunrpc unreachable for
200.204.153.224.4136 > my.net.215.62.sunrpc: [|tcp] (DF) (ttl 49, id 31435, len 60) (DF) [tos 0xc0]  (ttl 255,
id 3072, len 88)
```

### 1. Source of Trace:

The data was gathered from my home network using the sensor described in the introduction.

### 2. Detect was generated by:

Me visually scanning tcpdump output (tcpdump was limited to collecting activity directly targeting the sensor, so the amount of data was not substantial).

### 3. Probability the source address was spoofed:

It is unlikely that the source IP was spoofed, scans are of limited use unless data can be sent back to the person scanning.

### 4. Description of attack:

According to the arachNIDS database there is only one intrusion event that involves port 111/tcp:

• IDS428/portmap-listing-111 [*TCP any -> 111*]

(in other vulnerabilities databases it is referred to as: CAN-1999-0632 and 2001705). This detect indicates that a query was sent to the portmap daemon, requesting port information for RPC services.

### 5. Attack mechanism:

This individual is obviously performing reconnaissance. Inappropriately configured and secured RPC services account for a great many compromised UNIX/LINUX systems. Many RPC programs can be accessed to acquire additional information about a target system (e.g., what programs are running, which users are currently logged in, etc.). This scan is easy to perform using the rpcinfo command:

```
$ rpcinfo –p MY.NET.215.62
   program    vers    proto   port
   100000     2       tcp     111    portmapper
   100000     2       udp     111    portmapper
   100024     1       udp     32768  status
   100024     1       tcp     32768  status
```

Had I not been blocking port 111, the scanner would have revealed that I have a process listening on port 32768 (as well as the portmapper on port 111).

**6. Correlation:**

To verify that “rpcinfo –p” is what created the detect, I went to another LINUX box on another subnet and ran the rpcinfo command against the sensor. The results were identical.

**7. Evidence of active targeting:**

This type of reconnaissance is seldom run against a single machine. Normally the scanner would create a simple script that runs the rpcinfo command against entire networks. I have no reason to suspect that I was singled out.

**8. Severity:**

The severity of the attack is a –1.

(Criticality + Lethality) – (System Countermeasures + Network Countermeasures) = Severity

Criticality = 3. Had this dedicated sensor been part of an enterprise IDS system, a compromise would effectively blind us to other activity against our systems.
Lethality = 2. This is a probe only, however, success could have supplied valuable information that could have been used for focused attacks.
System Countermeasures = 4. With port 111 being blocked by a software firewall on the sensor, there is no chance of the probe succeeding.
Network Countermeasures = 2. Because of the number of exploits involving port 111, this type of activity is easily spotted by most signature-based IDS (and in my case, easily noted in the tcpdump output).

**9. Defensive recommendation:**

Though the firewall prevented this scan from succeeding, if at all possible, the portmap program should be turned off. If that is not possible, then a combination of firewall controlled access to port 111 (by specifying the hosts which are permitted to connect), and/or a version of portmap that support tcpwrappers should provide sufficient protection.

**10. Multiple choice test question:**

```
08:54:33.016263 200.204.153.224.4136 > my.net.215.62.111: S 1846790862:1846790862(0) win
32120 <mss 1460,sackOK,timestamp 15343585[|tcp]> (DF) (ttl 49, id 31435, len 60)
```

The tcpdump detected event above could be targeting which of the following operating systems:

a) Windows NT 3.51
b) Solaris 2.7
c) Windows 98
d) RedHat 7.1
e) b and d

answer is e

## Detect # 3 - Port 515 Scan

```
06:23:45.946263 211.227.49.4.4073 > my.net.215.62.printer: S 909841473:909841473(0) win 32120 <mss
1460,sackOK,timestamp 47402697[|tcp]> (DF) (ttl 48, id 59059, len 60)
06:23:46.076263 my.net.215.62 > 211.227.49.4: icmp: my.net.215.62 tcp port printer unreachable for
211.227.49.4.4073 > my.net.215.62.printer:
[|tcp] (DF) (ttl 48, id 59059, len 60) (DF) [tos 0xc0]  (ttl 255, id
2048, len 88)
```

**1. Source of Trace:**

The data was gathered from my home network using the sensor described in the introduction.

**2. Detect was generated by:**

Me visually scanning tcpdump output (tcpdump was limited to collecting activity directly targeting the sensor, so the amount of data was not substantial).

**3. Probability the source address was spoofed:**

It is unlikely that the source IP was spoofed, scans are of limited use unless data can be sent back to the person scanning.

**4. Description of attack:**

According to the arachNIDS database there are two intrusion events that involves port 515/tcp:

• IDS457/LPRng-redhat7-overflow-security.is [*TCP any -> 515*]
• IDS456/LPRng-redhat7-overflow-rdC [*TCP any -> 515*]

(in other vulnerabilities databases it is referred to as: CVE# CAN-2000-0917 and Bugtrag# 1711). Both events cover a vulnerability with the LPRng printing software. Though this software is available for multiple versions of UNIX/LINUX, RedHat 7.0 is particularly vulnerable, as it is installed by default.

**5. Attack mechanism:**

The LPRng exploit  involves a format string vulnerability in the use_syslog() function in LPRng 3.6.24. This allows remote attackers to execute arbitrary commands. Though this software is available for multiple versions of UNIX/LINUX, RedHat 7.0 is particularly vulnerable, as it is installed by default.

**6. Correlation:**

The information provided at www.whitehats.com is confirmed by RedHat security bulletin RHSA-2000-065-06 .

**7. Evidence of active targeting:**

This type of reconnaissance is seldom run against a single machine. I have no reason to suspect that I was singled out.

**8. Severity:**

The severity of the attack is a 2.

(Criticality + Lethality) – (System Countermeasures + Network Countermeasures) = Severity

Criticality = 3. Had this dedicated sensor been part of an enterprise IDS system, a compromise would effectively blind us to other activity against our systems.
Lethality = 5. Though this was only a probe, had it found that my 515 port was listening the exploit would have been attempted. Because the LPRng print software runs as a privledge user, administrator access would have been obtained.
System Countermeasures = 4. With port 515 being blocked by a software firewall on the sensor, there is no chance of the probe succeeding. Also, being aware of this exploit, I insured that my LPRng software was upgraded to the latest version which corrects the vulnerability.
Network Countermeasures = 2. Because of recent worm activity involving port 515, this type of activity is easily spotted by most signature-based IDS (and in my case, easily noted in the tcpdump output).

**9. Defensive recommendation:**

With all versions of UNIX, you should deactivate all unused services. If the machine will not be printing (such as with this sensor), then the print software should not be installed. Though I have the software installed, it is not running, and port 515 is blocked at the firewall.

**10. Multiple choice test question:**

Which of the following ports should be blocked at the network firewall to prevent the exploitation of the LPRng vulnerability:

a) port 53
b) port 80
c) port 111
d) port 515

The answer is d.

## Detect # 4 - Port 22 Attempt to Connect

```
19:56:44.760000 nl-ine-01.amst2.eu.psigh.com.ssh > my.net.215.62.ssh: S [tcp sum ok]
809740714:809740714(0) win 40 (ttl 28, id 39426, len 40)
19:56:44.760000 my.net.215.62.ssh > nl-ine-01.amst2.eu.psigh.com.ssh: S [tcp sum ok]
3467595606:3467595606(0) ack 809740715 win 5840 <mss 1460> (DF) (ttl 64, id 0, len 44)
19:56:45.000000 nl-ine-01.amst2.eu.psigh.com.ssh > my.net.215.62.ssh: R [tcp sum ok]
809740715:809740715(0) win 0 (ttl 241, id 46853, len 40)
19:56:45.170000 nl-ine-01.amst2.eu.psigh.com.2844 > my.net.215.62.ssh: S 3455662731:3455662731(0)
win 32120 <mss 1460,sackOK,timestamp 201938338[|tcp]> (DF) (ttl 50, id 46869, len 60)
19:56:45.170000 my.net.215.62.ssh > nl-ine-01.amst2.eu.psigh.com.2844: S 3455707127:3455707127(0)
ack 3455662732 win 5792 <mss 1460,sackOK,timestamp 113080459[|tcp]> (DF)(ttl 64, id 0, len 60)
```

```
19:56:45.410000 nl-ine-01.amst2.eu.psigh.com.2844 > my.net.215.62.ssh: . [tcp sum ok] 1:1(0) ack 1 win
32120 <nop,nop,timestamp 201938362 113080459> (DF) (ttl 50, id 46878, len 52)
19:56:45.430000 my.net.215.62.36394 > nl-ine-01.amst2.eu.psigh.com.finger: S
3455953442:3455953442(0) win 5840 <mss 1460,sackOK,timestamp 113080485[|tcp]> (DF) (ttl 64, id
53895, len 60)
19:56:45.670000 nl-ine-01.amst2.eu.psigh.com.finger > my.net.215.62.36394: R [tcp sum ok] 0:0(0) ack
3455953443 win 0 (ttl 241, id 46887, len 40)
19:56:45.690000 my.net.215.62.ssh > nl-ine-01.amst2.eu.psigh.com.2844: F [tcp sum ok] 1:1(0) ack 1 win
5792 <nop,nop,timestamp 113080511 201938362> (DF) (ttl 64, id 40795, len 52)
19:56:45.930000 nl-ine-01.amst2.eu.psigh.com.2844 > my.net.215.62.ssh: . [tcp sum ok] 1:1(0) ack 2 win
32120 <nop,nop,timestamp 201938414 113080511> (DF) (ttl 50, id 46900, len 52)
19:56:45.930000 nl-ine-01.amst2.eu.psigh.com.2844 > my.net.215.62.ssh: F [tcp sum ok] 1:1(0) ack 2 win
32120 <nop,nop,timestamp 201938414 113080511> (DF) (ttl 50, id 46902, len 52)
19:56:45.930000 my.net.215.62.ssh > nl-ine-01.amst2.eu.psigh.com.2844: . [tcp sum ok] 2:2(0) ack 2 win
5792 <nop,nop,timestamp 113080535 201938414> (DF) (ttl 255, id 0, len 52)
```

### 1. Source of Trace:

The data was gathered from my home network using the sensor described in the introduction.

### 2. Detect was generated by:

Me visually scanning tcpdump output (tcpdump was limited to collecting activity directly targeting the sensor, so the amount of data was not substantial).

### 3. Probability the source address was spoofed:

The source IP was not spoofed, as there was more than one successful three-way hand shake.

### 4. Description of attack:

This was an unauthorized access attempt, someone at amst2.eu.psigh.com attempted to log into the sensor using SSH. As specified in the introduction, port 22 (SSH) is the only open port on the sensor. It is possible that this was just scanning to find hosts with an open port 22, but normally scanners reset the connection after getting the initial response. This attempt appears to be continued reconnaissance, I don't think they intended to break in using a single login attempt. However, they may have been trying to find out what version of SSH I was running.

### 5. Attack mechanism:

Had the individual at amst2.eu.psigh.com known a valid account name, the password, or had their host been included in shosts.equiv file, they could have gained access to the sensor. As it was, they probably at most found out that I am not running an early version of OpenSSH which was reported to have an buffer overflow vulnerability.

### 6. Correlation:

I attempted to log into the sensor using a bogus account, and checked the tcpdump output. The results were identical.

### 7. Evidence of active targeting:

If this was not a scan, and I don't think is was for reasons outlined in section 4, then this was definitely a case of active targeting. It is possible that they performed a scan earlier to determine if port 22 was open, and this visit was to get the banner information from SSH.

**8. Severity:**

The severity of the attack is a -1.

(Criticality + Lethality) – (System Countermeasures + Network Countermeasures) = Severity

Criticality = 3. Had this dedicated sensor been part of an enterprise IDS system, a compromise would effectively blind us to other activity against our systems.
Lethality = 2. This was probably an attempt to find out what version of SSH I am running. As there are no known vulnerabilities with the version of SSH on the sensor, there is not much they could do to the sensor.
System Countermeasures = 5. Even though port 22 is not blocked, there is nothing an intruder could do to the sensor via this port. There are only two accounts on the sensor: root, which is restricted to the console, and an unprivileged account which can sudo to root (if they know the root password). The inbound use of SSH is restricted to a single machine, which sits behind a broadband router/firewall with all ports blocked.
Network Countermeasures = 1. Because this is actually normal activity, I don't think many IDS would complain. I noticed it because it was directed at the sensor.

**9. Defensive recommendation:**

The use of SSH is definitely preferred over telnet or rsh. However, if not properly configured, it can be as insecure as rsh with a "++" in the /etc/hosts.equiv. Properly configured, SSH can counter the following attacks:

Eavesdropping
Name Service and IP Spoofing
Connection Hijacking
Man-in-the-Middle Attacks
Insertion Attacks

For assistance in properly configuring SSH you should consult SSH The Secure Shell: The Definitive Guide, by Daniel J. Barrett & Richard E. Silverman (O'Reilly, 2001).

**10. Multiple choice test question:**

The advantages of using SSH is:

a) secure remote logins
b) secure file transfers
c) secure remote command execution
d) port forwarding
e) all the above

The answer is e.

## Detect # 5 - ASP (SubSeven v.2) Scan

```
21:24:42.940000 cc949996-a.indnpls1.in.home.com.2189 >
my.net.215.62.asp: S [tcp sum ok] 12254327:12254327(0) win 65535 <mss 1460,nop,nop,sackOK> (DF) (ttl
113, id 28175, len 48)
21:24:42.940000 my.net.215.62 > cc949996-a.indnpls1.in.home.com: icmp: my.net.215.62 tcp port asp
unreachable for cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp: [|tcp] (DF) (ttl 113, id 28175,
len 48) (DF) [tos 0xc0]  (ttl 255, id 26626, len 76)
21:24:45.810000 cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp: S [tcp sum ok]
12254327:12254327(0) win 65535 <mss 1460,nop,nop,sackOK> (DF) (ttl 113, id 42767, len 48)
21:24:45.810000 my.net.215.62 > cc949996-a.indnpls1.in.home.com: icmp:
my.net.215.62 tcp port asp unreachable for cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp:
[|tcp] (DF) (ttl 113, id 42767, len 48) (DF) [tos 0xc0]  (ttl 255, id 26882, len 76)
21:24:51.800000 cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp: S [tcp sum ok]
12254327:12254327(0) win 65535 <mss 1460,nop,nop,sackOK> (DF) (ttl 113, id 47375, len 48)
21:24:51.800000 my.net.215.62 > cc949996-a.indnpls1.in.home.com: icmp:
my.net.215.62 tcp port asp unreachable for cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp:
[|tcp] (DF) (ttl 113, id 47375, len 48) (DF) [tos 0xc0]  (ttl 255, id 27138, len 76)
21:25:03.810000 cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp: S [tcp sum ok]
12254327:12254327(0) win 65535 <mss 1460,nop,nop,sackOK> (DF) (ttl 113, id 9744, len 48)
21:25:03.810000 my.net.215.62 > cc949996-a.indnpls1.in.home.com: icmp:
my.net.215.62 tcp port asp unreachable for cc949996-a.indnpls1.in.home.com.2189 > my.net.215.62.asp:
[|tcp] (DF) (ttl 113, id 9744, len 48) (DF) [tos 0xc0]  (ttl 255, id 27394, len 76)
```

### 1. Source of Trace:

The data was gathered from my home network using the sensor described in the introduction.

### 2. Detect was generated by:

Me visually scanning tcpdump output (tcpdump was limited to collecting activity directly targeting the sensor, so the amount of data was not substantial).

### 3. Probability the source address was spoofed:

It is unlikely that the source IP was spoofed, scans are of limited use unless data can be sent back to the person scanning.

### 4. Description of attack:

The tcpdump output is a little confusing, even though it says the destination port was the ASP port, it actually refers to port 27374. On LINUX, when tcpdump fills in the name of a service, it gets that information from the /etc/services file. /etc/services lists port 27374 as being assigned to the Address Search Protocol, we know however that there are a number of vulnerabilities that target this port (none of which involve the Address Search Protocol). According to the arachNIDS database there are two intrusion events that involves port 27374/tcp:

• IDS461/worm-ramen-asp-retrieval-outgoing [*TCP any -> 27374*]
• IDS460/worm-ramen-asp-retrieval-incoming [*TCP any -> 27374*]

However, I don't believe this to be RAMEN worm activity. A much more serious threat utilizing port 27374 has been on the prowl for the last several months, SubSeven v.2. In fact, this type of scanning is so common that port 27374 is on Stephen Northcutt's list of top four ports to monitor. I was surprised that www.whitehats.com didn't have anything on this threat, but other security sites did:

NetworkIce
http://advice.networkice.com/advice/Phauna/RATs/SubSeven/default.htm
CVE CAN-1999-0660 A hacker utility or Trojan Horse is installed on a system.
CAN-2000-0138 A system has a distributed denial of service (DDOS) attack master, agent, or zombie installed.

Apparently, someone is searching for SubSeven v2 compromised hosts.

**5. Attack mechanism:**

Similar to the trojan documented in Detect #1, SubSeven can be installed via any number of ways. Unlike BackOrifice though, SubSeven usually checks in to a IRC server from where it can be controlled. Though it is normally controlled from the IRC server, there is a backdoor that listens on port 27374. An apparently popular activity in the Internet underworld is the hijacking of SubSeven infected systems. Since the release of the original SubSeven we have seen considerable scanning for these backdoors. If this were a SubSeven compromised hosts, and the password had not been changed, then the scanner would have unlimited access to the system.

**6. Correlation:**

Though I don't have access to SubSeven compromised boxes to verify what's been written about them being used to scan for other compromised hosts, observed scanning of this type, noticed at work, has led us to compromised boxes.

**7. Evidence of active targeting:**

This type of reconnaissance is seldom run against a single machine. I have no reason to suspect that I was singled out.

**8. Severity:**

The severity of the attack is a -2.

(Criticality + Lethality) – (System Countermeasures + Network Countermeasures) = Severity

Criticality = 3. Had this dedicated sensor been part of an enterprise IDS system, a compromise would effectively blind us to other activity against our systems.
Lethality = 1. Even if the scan had revealed that port 27374 was open, SubSeven only affects Microsoft Windows operating systems.
System Countermeasures = 4. As the scanner found out, multiple times, the firewall is effectively blocking port 27374.

Network Countermeasures = 2. Because of recent trojan activity involving port 27374, this type of activity is easily spotted by most signature-based IDS (and in my case, easily noted in the tcpdump output).

**9. Defensive recommendation:**

The recommendations I made for Detect #1 are basically true for any Microsoft-based trojans. If you are running any of the Microsoft operating systems, you need to block port 27374 at the network firewall, scan your system regularly using any of the standard anti-virus software packages (they all detect SubSeven v.1 & 2), and install a personal firewall that detect outbound as well as inbound Internet connections (such as ZoneAlarm) on each system. Another thing you can monitor is IRC activity, assuming your enterprise doesn't routinely use IRC, you should monitor for activity going to port 6667. When a system is compromised, and periodically after that, it will connect to an IRC server for instructions.

**10. Multiple choice test question:**

The netstat command is run on one of your Microsoft NT systems, and it shows a process listening on port 27374. What is this type of activity does this indicate?

a) SubSeven v.1 server
b) BackOrifice server
c) Doom server
d) SubSeven v.2 server

# GIAC Practical Assignment 2

# Passive OS Fingerprinting: A Serious Threat

## *Introduction*

Anyone who has spent any time in computer security knows the drill. First they probe your networks to see what systems are really there, they try to determine what services are offered and/or what operating system you are running, and then they start attempting to exploit your systems based on their reconnaissance. Granted there are some variations, many script kiddies just scan for certain ports (31337, 12345, etc) and then start their attacks. Fortunately, they tend to be very noisy and are easy to spot. The ones you have to watch out for are the quiet ones who find out what operating system you are running. Once they know your OS, they can take their pick of vulnerabilities (most of OS' have more than one) and using specialized scripts pop your box in seconds.

This paper will be a review of OS fingerprinting, with special emphasis on passive OS fingerprinting which I consider to be the greatest threat.

## *OS Fingerprinting: Why?*

Why would anyone care about what operating system you are running? Reconnaissance has always been the key in any military engagement, and computer security/cracking has many correlations with the military arts. You wouldn't send your troops over a hill without first knowing who/what is on the other side and what the terrain is like. The same is true for people who attack computer systems, they want to know what they are dealing with or their chances of success are slim. By figuring out what operating system you are running, they can then figure out what vulnerabilities you may be susceptible to.

## *OS Fingerprinting: Types*

In the early days of cracking, the only way to determine your prey's operating system was with banners. You would telnet to port 23 of the target box and get something like:

```
Escape character is '^]'.

HP-UX hpux B.10.01 A 9000/715 (ttyp2)

Login:
```

Or, you would ftp and get something like:

```
Trying 192.168.1.7 …
Connected to ftp.myhouse.com
Escape character is ‘^]’.
220 ftp29 FTP server (UNIX® System V Release 4.0) ready.
SYST
215 UNIX Type: L8 Version: SUNOS
```

This type of OS fingerprinting is considered active, which makes it detectable, and is not very efficient. Most OS fingerprinting tools today, derive their information from the TCP stack and how it is implemented. For the most part, each OS implements the TCP stack a little differently. By examining things like flag settings, Time-To-Live (TTL), window size, maximum segment size, don’t fragment flag, sackOK option, nop option, and window scaling option, the latest fingerprinting tools can achieve a high level of success. Some of the tools that use this method are nmap, queso, checkos, sirc, p0f, siphon, and many others. Though this method is not foolproof, it can achieve a success rating above ninety percent.

The other way fingerprinting tools are categorized are by how they get the TCP stack data. Active fingerprinters, such as nmap, query the stack, either directly for the information or by sending it bogus data and seeing how it reacts. This approach can lead to more accurate results, but is also easily noticed. SNORT (a popular signature-based intrusion detection application) has in it’s default rule set a rule for detecting NMAP OS fingerprinting. The other type of fingerprinters are passive, they acquire their TCP data from packets coming from the target host as a result of normal TCP traffic. This normal data can be from authorized users going about their daily tasks, such as that captured by sniffers or firewalls, or be the result of the attacker performing normal seeming tasks, such as visiting a website. Because passive fingerprinters never directly send packets to the target host, you will never know if you are being fingerprinted. For this reason, I considered passive OS fingerprinters to be the greatest threat.

## *OS Fingerprinting: p0f (an example of a passive fingerprinter)*

To better illustrate the threat of passive OS fingerprinters I decided to take a closer look at one, p0f . There are more well known fingerprinters than p0f, but they have already been documented in detail. Another reason I chose p0f is that it does an excellent job illustrating how simple a process it is to determine what OS you are running from just one TCP packet. I’m not implying that the coding task was easy, only that his utility’s code is succinct and to the point.

P0f was written by Michal Zalewski and the source code can be found at http://lcamtuf.coredump.cx/soft/p0f.tgz. I evaluated p0f version 1.7 under RedHat

Linux version 7.1. It compiled without incident, and installation consisted of placing the database file under /etc and the binary under /usr/local/bin (actually the binary can be placed anywhere in your path). P0f has the following options:

```
	usage: p0f [-f file][-i device][-s file][-v][‘filter rule’]
	-f file		read fingerprint information from file
	-i device	read packets from device
	-s file		read packets from device
	-v		verbose
```

p0f sends its output to standard out and is redirectable to a file or filter process using regular UNIX redirects and/or pipes. As warned by Mr. Zalewski in his readme file, p0f (like all TCP fingerprinters) can be adversely affected by firewalls and proxies.

Once it is working, p0f’s output appears as such:

```
[root@mysystem /]# ./p0f
p0f: passive os fingerprinting ver. 1.7 by <lcamtuf@tpi.pl>
p0f: file: '/etc/p0f.fp', 64 fprints, iface: 'eth0', rule: 'all'.
Kernel filter, protocol ALL, TURBO mode (671 frames), raw packet socket
XXX.XXX.XXX.XXX [7 hops]: Linux 2.2.12-20 (RH 6.1)
XXX.XXX.XXX.XXX [12 hops]: Solaris 2.6 (2)
XXX.XXX.XXX.XXX [9 hops]: Windows NT 4.0
XXX.XXX.XXX.XXX: UNKNOWN [5840:64:1544:1:0:1:1].
```

The XXXs are masked IP addresses, and the last entry was one of my RedHat 7.1 boxes. Upon examining the database file, I saw that RedHat 6.1 was the latest version of RedHat since the database file was created.

```
/etc/f0p.fp
31072:64:3884:1:0:1:1:Linux 2.2.12-20 (RH 6.1)
512:64:1460:0:0:0:0:Linux 2.0.38
32120:64:1460:1:0:1:1:Linux 2.2.14 or Cobalt Linux 2.2.12C3
16384:64:1460:1:0:0:0:FreeBSD 4.0-STABLE, 3.2-RELEASE
8760:64:1460:1:0:0:0:Solaris 2.6 (2)
9140:255:9140:1:0:0:0:Solaris 2.6 (sunsite)
49152:64:1460:0:0:0:0:IRIX 6.5 / 6.4
8760:255:1460:1:0:0:0:Solaris 2.6 or 2.7
8192:128:1460:1:0:0:0:Windows NT 4.0
8192:128:1460:1:0:1:1:Windows 9x (1)
8192:128:536:1:0:1:1:Windows 9x (2)
2144:64:536:1:0:1:1:Windows 9x (4)
16384:128:1460:1:0:1:1:Windows 2000
32120:32:1460:1:0:1:1:Linux 2.2.13
8192:32:1460:1:0:0:0:Windows NT 4.0
5840:128:536:1:0:1:1:Windows 95 (3)
.
.
.
```

Fortunately for the user, updating the database file consists of taking the data in brackets and adding it to the database along with identification information.

```
5840:64:1544:1:0:1:1:Linux 2.4.2-2 (RH 7.1)
```

Upon adding the new entry and attempting to connect to a non-existing web server on the p0f machine, I get:

```
XXX.XXX.XXX.XXX [1 hops]: Linux 2.4.2-2 (RH 7.1)
```

Besides getting the data straight from the pipe, you can also use P0f using data files. The only ones I had on hand were some tcpdump files, which worked fine. It also has the capability to use 'filter rules'. I tried it with a simple rule "host XXX.XXX.XXX.XXX" and it seemed to work fine.

## *OS Fingerprinting: Uses*

Some in the cracking community would have us believe that passive OS fingerprinters have a lot of uses for the computer security industry. The only one that is even close to being believable is when the Honeynet Project talks about it being used to identify 'rogue' systems on your network. The principle being, if you have a MS Windows and Sun Solaris based network, checking all outbound IPs might reveal illegal LINUX or FreeBSD machines. Personally, there is only one use for OS fingerprinters, reconnaissance.

## *OS Fingerprinting: Defending Against*

There isn't a whole lot that your average system administrator or system security personnel can do about passive OS fingerprinting. Firewalls and proxies can limit the amount of packets returned from the actual machine. And some have come up with innovative ways to modify your TCP stack, such as Gaël Roualland and Jean-Marc Saffroy's IP Personality (found at http://ippersonality.sourceforge.net), which allows you to change the TCP Initial Sequence Number, Window Size, and TCP options on Linux 2.4.0 kernels. But these are not feasible solutions for most system administration or security staffs. The only real way to mitigate the threat posed by any OS fingerprinter, is to assume that everyone already knows what operating systems you are using and to keep your systems patched.

## *References:*

Zalewski, Michal. "p.0.f Readme file." URL: http://lcamtuf.coredump.cx/soft/p0f.tgz (7 July 2001)

Fyodor. "Remote OS Detection via TCP/IP Stack FingerPrinting." 18 October 1998. URL: http://www.insecure.org/nmap/nmap-fingerprinting-article.html 12 July 2001.

Honeynet Project. “Know Your Enemy: Passive Fingerprinting.” 24 May 2000
URL: http://project.honeynet.org 27 June 2001.

Roualland, Gaël and Saffroy, Jean-Marc. “IP Personality.” URL:
http://ippersonality.sourceforge.net 8 June 2001.

## Appendix A. p0f Readme.txt

--=--
p.0.f
--=--

"Dr. Jekyll had something to Hyde"

passive OS fingerprinting tool
version 1.7 <lcamtuf@tpi.pl>

-= buffer0verfl0w security team =-

http://lcamtuf.hack.pl/p0f.tgz

Special thanks to:

* Lance Spitzner for whitepaper on passive OS fingerprinting:
  http://www.enteract.com/~lspitz/finger.html
* tf8 for initial piece of libpcap support and packet parsing
* teso/security.is/b0f/#hax for ideas and testing
* Jeremy Weatherford, Chris Wilson and Szilveszter Adam for
  portability testing/patches, bugfixes and ideas,
* other BUGTRAQ readers for OS fingerprints and useful patches
* other people involved (or not) in this project
* very, very special thanks to el- :*

Background:

* What is passive OS fingerprinting?

Passive OS fingerprinting technique bases on information coming from remote host when it establishes connection to our system. Captured packets contains enough information to determine OS - and, unlike active scanners (nmap, queSO) - without sending anything to this host.

If you're looking for more information, read Spitzner's text at:
http://www.enteract.com/~lspitz/finger.html

* How it works?

Well, there are some TCP/IP flag settings specific for given systems. Usually initial TTL (8 bits), window size (16 bits), maximum segment size (16 bits), don't fragment flag (1 bit), sackOK option (1 bit), nop option (1 bit) and window scaling option (8 bits) combined together gives unique, 51-bit signature for every system.

* What are main advantages?

Passive OS fingerprinting can be done on huge portions of input data - eg. information gathered on firewall, proxy, routing device or Internet server, without causing any network activity. You can launch passive OS detection software on such machine and leave it for days, weeks or months, collecting really interesting statistical and - *erm* - just interesting information. What's really funny - packet filtering firewalls, network address translation and so on are transparent to p0f-alike software, so you're able to obtain information about systems behind the firewall. Also, such software can determine distance between remote host and your system, allowing you to generate network structure maps for firewalled/structural networks. And all without sending _any_ packet. Just think about IDS systems.

Limitations

Proxy firewalls and other high-level proxy devices are not transparent to any tcp fingerprinting software. It applies to p0f, as well.

In order to obtain information required for fingerprinting, you have to receive at least one SYN packet initializing TCP connection to your machine or network. Note: you don't have to respond to particular SYN. Of course, it's impossible to perform any kind of OS detection witout receiving any information.

It is possible to perform fingerprinting on alive TCP connection or even when connection is initialized from your network. But these techniques are less realible (as in many implementations some parameters are copied from first SYN packet, so if connection has been initialized from our network, fingerprinting won't be successful; also, some parameters like window size are constant for initial TCP/IP packet, but changing rapidly later).

Why our bubble gum is better?

There is another passive OS detection utility, called 'siphon'. It's pretty good piece of proof-of-concept software, but it isn't perfect. Well, p0f isn't perfect for sure, but has several improvements:

- it's single-threaded and pretty clean,

- works properly on Linuxes (siphon has a problem with bpf on 2.2), as well as on BSD systems and SunOS/Solaris,

- has pretty large and detailed fingerprints database,

- uses more information for fingerprinting (26 extra bits),

- it's more accurate,

- you can define your own filtering rules in the tcpdump flavour: p0f 'src host 1.2.3.4' or p0f 'gateway 1.2.3.4 and port 80', and listening interface (using option -i).

What more? Dunno :) Simply, check it out.

Not working!

Probably p0f isn't working well on every platform in the world; first of all, you'll need libpcap 0.4 or newer; sometimes pcap.h is placed in /usr/include/pcap instead of /usr/include/ (eg. in broken RH 6.1 package). In this case, simply issue:

```
ln -s /usr/include/pcap/pcap.h /usr/include/
ln -s /usr/include/pcap/net/bsf.h /usr/include/net/
```

NOTE: if p0f recognized system incorrectly or cannot recognize it at all, please send OS signature and system description to author. Thanks :)

Tested platforms:

- NetBSD
- FreeBSD
- OpenBSD
- Linux 2.0/2.2
- Solaris 2.6-2.7

Requires: libpcap 0.4 or newer; GNU cc 2.7.x or newer; GNU make 3.7x;
GNU egrep (for proper Makefile processing)

Files:

/etc/p0f.fp or ./p0f.fp - OS fingerprints database. Format is described inside:

```
# Valid entry describes the way server starts TCP handshake (first SYN).
# Important options are: window size (wss), maximum segment size (mss),
# don't fragment flag (DF), window scaling (wscale), sackOK flag, nop
# flag, and initial time to live (TTL) ;)
#
# How can you determine initial ttl? Well, usually it's first power of 2
# bigger than TTL returned in scan. So, for example, if you get TTL 55 in
# fingerprint returned by p0f, initial TTL will be usually 64... NOTE:
# it's better to overestimate initial TTL than underestimate it ;)
#
# There are some brain-damaged devices, like network printers etc, that
# have stupid initial TTLs like 60, but who cares, if HP LaserJet wants to
# visit your server, you have to think again about your life ;)
#
# Format:
#
# wwww:ttt:mmm:D:W:S:N:OS Description
#
# wwww - window size
# ttt  - time to live
# mmm  - maximum segment size
# D    - don't fragment flag  (0=unset, 1=set)
# W    - window scaling (-1=not present, other=value)
# S    - sackOK flag (0=unset, 1=set)
# N    - nop flag (0=unset, 1=set)
```

What to do?

- COLORFUL INTERFACE :))))

License, disclaimer:

The p0f utility and related utilities are free software; you can redistribute it and/or modify it under the terms of the GNU Library General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL MICHAL ZALEWSKI, OR ANY OTHER CONTRIBUTORS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

- Michal Zalewski lcamtuf@tpi.pl

## Appendix B. p0f.c

```
/*

  p0f - passive OS fingerprinting
  -------------------------------
  (c) <lcamtuf@tpi.pl>

  The p0f utility and related utilities are free software; you can redistribute it and/or modify it under
the terms of the GNU Library General Public License as published by the Free Software
Foundation; either version 2 of the License, or (at your option) any later version.

  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL MICHAL ZALEWSKI, OR ANY OTHER CONTRIBUTORS BE LIABLE
FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF
CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH
THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

*/

#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <pcap.h>
#include <arpa/inet.h>
#include <signal.h>
#include <unistd.h>

#include "tcp.h"
#define MAXFPS 1000
#define FPBUF  120
#define INBUF  1024
#define TTLDW  30

#ifndef VER
#  define VER "(?)"
#endif /* !VER */

extern char *optarg;
extern int optind;

char fps[MAXFPS][FPBUF];
int wss, wscale, mss, nop, ttl, df, sok,tmp,header_len=14,dupa;
int verbose=0,sp,dp;
struct in_addr sip,dip;
struct bpf_program flt;
pcap_t *pt;

void die_nicely() {
  pcap_close(pt);
  exit(0);
}

void lookup(void);

void parse(u_char *blabla, struct pcap_pkthdr *pph, u_char *packet) {
  struct iphdr *iph;
  struct tcphdr *tcph;
  int ilen=0, hlen=0,off,olen;
  dupa=0;

  if (pph->len < header_len+sizeof(struct iphdr)+sizeof(struct tcphdr)) {
    return;
  }
  // Rare tropical disease ugly dirty obfuscated hack ;>
  iph=(struct iphdr*) (packet);
  if ((iph->ihl>>4)!=4 || iph->protocol!=IPPROTO_TCP)
    iph=(struct iphdr*)(packet+header_len);
  if ((iph->ihl>>4)!=4 || iph->protocol!=IPPROTO_TCP) {
    int a,b;
    iph=(struct iphdr*) (packet);
    // Change ihl byteorder, endian detection ;)
    a=iph->ihl&15;b=(iph->ihl>>4)&15;iph->ihl=a*16+b;
    if ((iph->ihl>>4)!=4 || iph->protocol!=IPPROTO_TCP)
      iph=(struct iphdr*)(packet+header_len);
    if ((iph->ihl>>4)!=4 || iph->protocol!=IPPROTO_TCP) {
      return;
    }
  }

  ttl=iph->ttl;

  off=ntohs(iph->off);
  df=((off&IP_DF)!=0);
  sip.s_addr=iph->saddr;
  dip.s_addr=iph->daddr;
  ilen= ( (iph->ihl&0x0f) );

  switch (ilen) {
    case 5: /* no options */
      tcph=(struct tcphdr *)(iph+1);
      break;
    default: /* parse ipoptions */
      if ((header_len+(ilen<<2)+sizeof(struct tcphdr)) > pph->len) {
            return;
      }
      tcph=(struct tcphdr *)(packet+header_len+(ilen<<2));
      break;
  }

  off=tcph->th_flags;
  if (!(off&TH_SYN)) return;
  if ((off&TH_ACK)) return;

  wscale=-1;
  mss=0;
  nop=0;
  sok=0;

  hlen=(tcph->th_off)*4;

  {
    void* opt_ptr;
    int opt;
    opt_ptr=(void*)tcph+sizeof(struct tcphdr);
    while (dupa<hlen) {
      opt=(int)(*(u_char*)(opt_ptr+dupa));
      dupa+=1;
      switch(opt) {
        case TCPOPT_EOL:
               dupa=100000; break; // Abandon ship!
        case TCPOPT_NOP:
               nop=1;
               break;
             case TCPOPT_SACKOK:
               sok=1;
               dupa++;
               break;
             // Long options....
             case TCPOPT_MAXSEG:
               dupa++;
               mss=EXTRACT_16BITS(opt_ptr+dupa);
               dupa+=2;
               break;
             case TCPOPT_WSCALE:
               olen=(int)*((char*)opt_ptr+dupa)-2; dupa++;
               if (olen<0) olen=0;
               wscale=(int)*((u_char*)opt_ptr+dupa);
               dupa+=olen;
               break;
             case TCPOPT_TIMESTAMP:
               olen=(int)*((char*)opt_ptr+dupa)-2; dupa++;
               if (olen<0) olen=0;
               dupa+=olen;
               break;
             default:
               olen=(int)*((char*)opt_ptr+dupa)-2; dupa++;
               if (olen<0) olen=0;
               dupa+=olen;
              break;
      }
    }
  }
#if BYTE_ORDER == LITTLE_ENDIAN
  sp=htons(tcph->th_sport);
  dp=htons(tcph->th_dport);
  wss=htons(tcph->th_win);
#else
  sp=tcph->th_sport;
  dp=tcph->th_dport;
  wss=tcph->th_win;
#endif
```

```
  lookup();
  return;
}

void lookup(void) {
  int i=0,got=0,down=0;
  int origw=wscale;
  char buf[INBUF],*p;
  char* plonked="\n";
plonk:
  for (down=0;down<TTLDW;down++) {
    i=0;
    sprintf(buf,"%d:%d:%d:%d:%d:%d:%d:",wss,ttl+down,mss,df,wscale,sok,nop);
    while (fps[i][0]) {
      if (!strncmp(buf, fps[i], strlen(buf))) {
        got=1;
        p=strrchr(fps[i],':')+1;
        if (strchr(p, '\n')) p[strlen(p)-1]=0;
        printf("%s [%d hops]: %s%s",inet_ntoa(sip),down+1,p,plonked);
             if (verbose) {
               printf(" + %s:%d ->",inet_ntoa(sip),sp);
               printf(" %s:%d\n", inet_ntoa(dip),dp);
             }
        break;
      }
      i++;
    }
    if (got) break;
  }
  if (!got) if (wscale==-1) { plonked=" *\n";wscale=0; goto plonk; }
  if (!got) printf("%s: UNKNOWN [%d:%d:%d:%d:%d:%d:%d].\n",
           inet_ntoa(sip), wss, ttl, mss, df, origw, sok, nop);
  fflush(0);
}

int fips;

void load_fprints(char *filename) {
  FILE *x;
  int i=0;
  char *p;
  x=fopen(filename, "r");
  if (!x) x=fopen("p0f.fp", "r");
  if (!x) {
    fprintf(stderr, "No OS fingerprint database (%s) found. Dumb mode on.\n",
      filename);
    return;
  }
  while (fgets(fps[i],FPBUF-1,x)) {
    if ((p=strchr(fps[i],'#')))            *p=0;
    if (fps[i][0]) i++;
  }
  fips=i;
  fclose(x);
}
```

```
char *ifa,*rul;

void usage(char* what) {
  fprintf(stderr,"p0f: %s\n",what);
  fprintf(stderr,"usage: p0f [ -f file ] [ -i device ] [ -s file ] [ -v ][ 'filter rule' ]\n");
  fprintf(stderr, " -f file   read fingerprint information from file\n");
  fprintf(stderr, " -i device read packets from device\n");
  fprintf(stderr, " -s file   read packets from file\n");
  fprintf(stderr, " -v        verbose mode\n");
  exit(1);
}

int main(int argc, char *argv[]) {
  char errbuf[PCAP_ERRBUF_SIZE];
  char *filename = NULL, *inputfile = NULL;
  int r, s = 0;

  while ((r = getopt(argc, argv, "f:i:s:v")) != -1) {
    switch (r) {
      case 'f':
        filename = optarg;
            break;
      case 'i':
            ifa = optarg;
            break;
      case 's':
        s = 1;
            inputfile = optarg;
            break;
      case 'v':
        verbose = 1;
            break;
      default:
            usage("Unknown option.");
    }
  }

  /* set a reasonable default fingerprint file */
  if (!filename || !*filename)
    filename = "/etc/p0f.fp";

  /* anything left after getopt'ing is a rule */
  if (argv[optind] && *(argv[optind]))
    rul = argv[optind];

  if (!ifa) ifa=pcap_lookupdev(errbuf);
  if (!ifa) { ifa="lo"; }

  fprintf(stderr, "p0f: passive os fingerprinting ver. " VER " by <lcamtuf@tpi.pl>\n");

  if (s && inputfile && *inputfile) {
    if ((pt=pcap_open_offline(inputfile, errbuf))==NULL) {
      fprintf(stderr, "pcap_open_offline failed: %s\n", errbuf);
      exit(1);
    }
  } else {
    if ((pt=pcap_open_live(ifa,100,1,100,errbuf))==NULL) {
      fprintf(stderr, "pcap_open_live failed: %s\n", errbuf);
      exit(1);
    }
  }

  signal(SIGINT,&die_nicely);
  signal(SIGTERM,&die_nicely);
  load_fprints(filename);

  if (pcap_compile(pt, &flt, rul?rul:"", 1, 0)) {
    if (rul) {
      pcap_perror(pt,"pcap_compile");
      exit(1);
    }
  }

  if (!rul) rul="all";
  fprintf(stderr,"p0f: file: '%s', %d fprints, iface: '%s', rule: '%s'.\n",filename,fips,ifa,rul);

  pcap_setfilter(pt, &flt);

  pcap_loop(pt,-1,(pcap_handler)&parse,(void*)0L);
  return 0; //not reached;>
}
```

# GIAC Practical Assignment 3

# Analyze This

This report is a security analysis of your site, as represented by the SNORT alert, scan, and oos (out of spec) files provided by you. In performing the analysis, I looked at seven days (3 Apr 2001 – 9 Apr 2001) worth of data, which came to just over 26 MB. By the end of this report you should have a better understanding of the security threats your networks face, as well as some ways you can mitigate them. This analysis will not identify all your network and system vulnerabilities, only the problems detected by SNORT for the period covered by the analysis. For a thorough security evaluation you should have a knowledgeable person or team perform a network and system vulnerability assessment.

SNORT is a signature-based intrusion detection system. Because it is signature-based, much of this analysis will revolve around the signature-based alerts reported by SNORT. I will:

- identify the signatures that SNORT matched while processing your network traffic
- identify the systems involved
- explain the threat
- suggest some ways you can mitigate the threat

NOTE: Because I don't have access to the actual SNORT rules that were used to evaluate your network traffic, there may be times that I have to speculate why SNORT thought there was a problem. There may also be times that I choose to limit to the top ten the list of sources and/or destinations. This will only occur when there are a substantial number of systems involved and in doing so does not detract from the analysis.

After covering the alerts I will go over the scan and oos data files, correlating them with the alerts where possible. I will then summarize the threats and provide a list of top ten sources and destinations.

The tools and processes used to perform this analysis will be thoroughly explained at the end of this report. The results of my analysis are as follows:

## *Timeframe of analysis*

Earliest alert at 00:00:04.56 on 04/03/2001
Latest alert at 23:49:57.69 on 04/09/2001

**Snort Alert Data Summary (73051 alerts recorded)**

| *Signature* | *# Alerts* | *# Sources* | *# Destinations* |
|---|---|---|---|
| STATDX UDP attack | 1 | 1 | 1 |
| Probable NMAP fingerprint attempt | 4 | 1 | 4 |
| connect to 515 from inside | 4 | 2 | 2 |
| ICMP SRC and DST outside network | 13 | 6 | 8 |
| Back Orifice | 16 | 1 | 16 |
| Port 55850 tcp Possible myserver activity | 32 | 15 | 19 |
| Tiny Fragments Possible Hostile Activity | 38 | 2 | 11 |
| High port 65535 udp possible Red Worm – traffic | 38 | 21 | 19 |
| Null scan! | 61 | 20 | 21 |
| NMAP TCP ping! | 63 | 8 | 46 |
| Queso fingerprint | 71 | 19 | 30 |
| TCP SRC and DST outside network | 104 | 24 | 49 |
| Watchlist 000222 NET-NCFC | 106 | 12 | 13 |
| SMB Name Wildcard | 149 | 99 | 90 |
| SUNRPC highport access! | 282 | 6 | 6 |
| UDP SRC and DST outside network | 504 | 46 | 274 |
| External RPC call | 513 | 15 | 411 |
| connect to 515 from outside | 819 | 19 | 549 |
| WinGate 1080 Attempt | 2802 | 79 | 2441 |
| SYN-FIN scan! | 2849 | 2 | 2693 |
| Possible RAMEN server activity | 4994 | 820 | 3439 |
| Attempted Sun RPC high port access | 5177 | 1 | 1 |
| High port 65535 tcp possible Red Worm – traffic | 6973 | 17 | 5459 |
| Watchlist 000220 IL-ISDNNET-990517 | 10144 | 41 | 36 |
| Possible trojan server activity | 11280 | 1327 | 7814 |
| Russia Dynamo - SANS Flash 28-jul-00 | 26014 | 4 | 4 |

## STATDX UDP attack ( 1 source, 1 destination )

Earliest such alert at 08:34:23.517421 on 04/07/2001
Latest such alert at 08:34:23.517421 on 04/07/2001

| Source | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 212.131.172.130 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.6.15 | 1 | 2 | 1 | 2 |

With certain types of LINUX there exists a vulnerability in the rpc.statd program which is part of the nfs-utils packages. Successfully exploited, this vulnerability allows a remote user to execute code as root.

Unfortunately, without the packet payload it is impossible to confirm this event, though the way most SNORT rules for STATDX-related events trigger is partially based off a string match in the payload.

If the destination machine is not a LINUX box, then this is probably a false alarm. If it is a LINUX box, it should be immediately examined to determine if it has been compromised. Normally after gaining root access the next step is to install a “root kit”, pre-determined software that allows increased control of the machine. Unfortunately, due to the sizeable number of “root kits” available, it is beyond the scope of this analysis to tell you what to look for to determine if your machine has been compromised. The examination should be performed by a knowledgeable individual, or if you want to be safe, reloaded from the original release media (don’t forget to apply all the appropriate patches).

For more information on this vulnerability see:
CVE-2000-0666, bugtraq#1480, advICE#2001702, or aracnid#442

## Probable NMAP fingerprint attempt (1 source, 4 destinations )

Earliest such alert at 07:30:13.573522 on 04/03/2001
Latest such alert at 08:09:04.359396 on 04/03/2001

| Source | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 217.3.182.110 | 4 | 57 | 4 | 41 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.53.114 | 1 | 3 | 1 | 1 |
| MY.NET.53.193 | 1 | 4 | 1 | 1 |
| MY.NET.53.29 | 1 | 2 | 1 | 1 |
| MY.NET.53.8 | 1 | 2 | 1 | 1 |

NMAP is a popular tool for remotely collecting information about a computer. Though used by a great many network/system security personnel to determine how others see their network and computers, it is also a popular recon tool for those who would like to gain unauthorized access to your systems. One of the features of NMAP is the ability to determine a machine's operating system (OS fingerprinting) and what services are available by using specially crafted packets.

The fact that these are occurrences of NMAP fingerprinting is confirmed by entries in the corresponding SNORT OOS file:

```
04/03-07:30:03.678998 217.3.182.110:43183 -> MY.NET.53.8:23
TCP TTL:31 TOS:0x0 ID:26319
**SF*P*U Seq: 0x302D88CF   Ack: 0x0   Win: 0xC00
TCP Options => WS: 10 NOP MSS: 265 TS: 1061109567 0 EOL EOL
```

The bolded characters represent flags that are available for use by TCP packets. The fact that the SYN, FIN, PUSH, and URGENT flags are all set is unusual (because it violates RFC ####, this combination of flags will never occur during normal TCP communications), and this combination matches packets crafted by NMAP.

This type of scanning poses a problem only if the destinations hosts are running TELNET (telnet listens on port 23 and that is where these probes were directed). If they are, then the individual who initiated the probe from the source listed above now knows of a way to access the probed systems. These systems should be watched for unauthorized login attempts. Even though there are TELNET vulnerabilities affecting certain operating systems, the biggest threat comes from unauthorized login attempts, and possible sniffer interception of authorized login sessions.

For more information on this type of probing see:
CVE CAN#1999-0454, advICE#2000314, or aracnid#5

## Connect to 515 from inside ( 2 sources, 2 destinations )

4 alerts with this signature

Earliest such alert at 10:01:58.095769 on 04/07/2001
Latest such alert at 21:45:37.823890 on 04/08/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.6.7 | 3 | 3 | 1 | 1 |
| MY.NET.206.146 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 128.8.3.103 | 3 | 3 | 1 | 1 |
| 207.226.225.3 | 1 | 1 | 1 | 1 |

This may or may not be a problem, depending on whether or not the two destinations are valid print servers. Port 515 is normally used by LPR the UNIX/LINUX Print Service. There have been reported exploits involving LPR, so this could also be attempts by the two sources to compromise the destination systems. The sources should be checked for signs of having been compromised. If they haven't been compromised, make sure that the owners of the destination machines don't mind you using their print servers.

### ICMP SRC and DST outside network ( 6 sources, 8 destinations )

13 alerts with this signature

Earliest such alert at 05:40:55.810786 on 04/03/2001
Latest such alert at 20:41:07.902709 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 172.129.5.60 | 4 | 4 | 1 | 1 |
| 172.168.1.242 | 4 | 4 | 2 | 2 |
| 169.254.101.152 | 2 | 53 | 2 | 30 |
| 172.175.71.210 | 1 | 1 | 1 | 1 |
| 172.168.1.9 | 1 | 1 | 1 | 1 |
| 172.148.38.182 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 198.207.223.246 | 4 | 4 | 1 | 1 |
| 206.196.158.113 | 3 | 3 | 1 | 1 |
| 172.132.23.250 | 1 | 1 | 1 | 1 |
| 210.149.145.60 | 1 | 1 | 1 | 1 |
| 62.254.55.241 | 1 | 1 | 1 | 1 |
| 208.57.0.134 | 1 | 1 | 1 | 1 |
| 203.41.198.130 | 1 | 1 | 1 | 1 |
| 168.187.25.237 | 1 | 1 | 1 | 1 |

This type of activity could be the results of a number of situations: 1) this traffic could be passing through because your network was determined to be the shortest route, or 2) someone in your network is using IP addresses that don't belong to them. As open as your network is, situation 1 would not surprise me. To prevent it you should tighten down your network (see my security advice comments at the end of this document). If it is situation two then this could cause serious problems for the real owners of those IPs. Verify that you are only using the IPs assigned to your organization.

### Back Orifice ( 1 source, 16 destinations )

16 alerts with this signature

Earliest such alert at 22:59:22.068369 on 04/06/2001
Latest such alert at 22:59:34.777449 on 04/06/2001

| Source | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 203.133.252.164 | 16 | 16 | 16 | 16 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.98.1 | 1 | 1 | 1 | 1 |
| MY.NET.98.106 | 1 | 1 | 1 | 1 |
| MY.NET.98.121 | 1 | 1 | 1 | 1 |
| MY.NET.98.128 | 1 | 1 | 1 | 1 |
| MY.NET.98.218 | 1 | 2 | 1 | 2 |
| MY.NET.98.154 | 1 | 1 | 1 | 1 |
| MY.NET.98.232 | 1 | 2 | 1 | 2 |
| MY.NET.98.166 | 1 | 1 | 1 | 1 |
| MY.NET.98.168 | 1 | 1 | 1 | 1 |
| MY.NET.98.244 | 1 | 2 | 1 | 2 |
| MY.NET.98.249 | 1 | 1 | 1 | 1 |
| MY.NET.98.79 | 1 | 1 | 1 | 1 |
| MY.NET.98.251 | 1 | 1 | 1 | 1 |
| MY.NET.98.83 | 1 | 1 | 1 | 1 |
| MY.NET.98.188 | 1 | 2 | 1 | 2 |
| MY.NET.98.197 | 1 | 1 | 1 | 1 |

Back Orifice is a Trojan that affects Microsoft Windows machines. It gives the attacker complete control of the compromised system. This was probably a scan for machines with processes listening on port 31337. If any of the destination machines are running Microsoft Windows, you could have a problem. You should go to them and run the netstat command, if there is a process listening on port 31337, then that box should be reloaded from the original CDs. There are also many commercial security products that can scan for the presence of Back Orifice, but to my knowledge (because of the multitude of tweaked versions of Back Orifice) there is no easy way to remove it.

For more information on this vulnerability see:
aracnid#188

## Port 55850 tcp - Possible myserver activity ( 15 sources, 19 destinations )

32 alerts with this signature

Earliest such alert at 04:16:21.818670 on 04/03/2001
Latest such alert at 18:56:24.817191 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.253.24 | 5 | 6 | 3 | 4 |
| MY.NET.6.35 | 4 | 7 | 2 | 5 |
| MY.NET.253.51 | 4 | 9 | 1 | 2 |
| 24.132.78.145 | 3 | 3 | 1 | 1 |
| MY.NET.202.2 | 2 | 2 | 1 | 1 |
| MY.NET.100.230 | 2 | 2 | 2 | 2 |
| 204.253.105.63 | 2 | 2 | 1 | 1 |
| 193.237.19.97 | 2 | 2 | 1 | 1 |
| 204.160.241.38 | 2 | 2 | 1 | 1 |
| 209.255.208.60 | 1 | 1 | 1 | 1 |
| MY.NET.217.174 | 1 | 2 | 1 | 2 |
| 172.151.3.65 | 1 | 1 | 1 | 1 |
| 212.171.9.23 | 1 | 1 | 1 | 1 |
| MY.NET.6.34 | 1 | 4 | 1 | 3 |

| | | | | |
|---|---|---|---|---|
| 216.33.35.214 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 152.163.224.88 | 4 | 4 | 1 | 1 |
| 204.253.105.63 | 3 | 3 | 1 | 1 |
| MY.NET.217.174 | 3 | 3 | 1 | 1 |
| 172.151.3.65 | 2 | 2 | 1 | 1 |
| 128.100.132.17 | 2 | 2 | 1 | 1 |
| MY.NET.222.76 | 2 | 3 | 1 | 2 |
| 207.172.4.98 | 2 | 2 | 1 | 1 |
| MY.NET.6.35 | 2 | 16 | 1 | 5 |
| MY.NET.100.230 | 2 | 5 | 1 | 4 |
| MY.NET.1.8 | 1 | 6 | 1 | 3 |
| 199.182.120.67 | 1 | 1 | 1 | 1 |
| 209.83.143.146 | 1 | 1 | 1 | 1 |
| MY.NET.224.150 | 1 | 1 | 1 | 1 |
| MY.NET.204.22 | 1 | 8 | 1 | 4 |
| 206.132.166.27 | 1 | 1 | 1 | 1 |
| 209.192.217.4 | 1 | 1 | 1 | 1 |
| 204.160.241.38 | 1 | 1 | 1 | 1 |
| MY.NET.202.2 | 1 | 1 | 1 | 1 |
| 24.132.78.145 | 1 | 1 | 1 | 1 |

This SNORT rule identifies traffic targeting and/or originating port 55850. Unfortunately, I am unsure why this is significant. This rule does not exist in the current SNORT rules 1.7, nor does a search of the primary security sites (www.incidents.org, www.sans.org, www.whitehats.com, etc.) or the CVE database reveal why we should be interested in port 55850. This could be a site-unique rule, related to unusual activity seen at your site involving this port. To be safe I would check the MY.NET hosts listed in the Sources section above for signs of compromise.

**Tiny Fragments Possible Hostile Activity ( 2 sources, 11 destinations )**

38 alerts with this signature

Earliest such alert at 11:20:39.466892 on 04/03/2001
Latest such alert at 15:48:21.112722 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 199.104.118.50 | 25 | 25 | 1 | 1 |
| 202.39.78.124 | 13 | 13 | 10 | 10 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.221.186 | 25 | 25 | 1 | 1 |
| MY.NET.211.222 | 2 | 2 | 1 | 1 |
| MY.NET.204.130 | 2 | 2 | 1 | 1 |
| MY.NET.207.50 | 2 | 4 | 1 | 3 |
| MY.NET.209.118 | 1 | 1 | 1 | 1 |
| MY.NET.207.106 | 1 | 1 | 1 | 1 |
| MY.NET.227.86 | 1 | 1 | 1 | 1 |
| MY.NET.214.134 | 1 | 1 | 1 | 1 |
| MY.NET.217.102 | 1 | 1 | 1 | 1 |
| MY.NET.209.42 | 1 | 1 | 1 | 1 |
| MY.NET.224.210 | 1 | 1 | 1 | 1 |

Fragmentation is a normal part of networking, it occurs whenever packets are too large for the network media. Sometimes however, it can also be used to bypass ID systems. Though there are reports of overlapping fragmented packets crashing certain operating systems (http://www.sans.org/newlook/resources/IDFAQ/fragments.htm), chances are the reported activity is a combination of scanning and normal fragmentation. The fact that you have SNORT alerts on this type of traffic is good, that means that fragmented packets can't be used to map your network without your knowledge. Just to be safe, you should insure that your systems are patched to handle attacks that utilize fragmented packets (Ping 'O Death and Teardrop).

## High port 65535 udp possible Red Worm – traffic (21 sources. 19 destinations)

8 alerts with this signature

Earliest such alert at 06:18:06.330076 on 04/04/2001
Latest such alert at 22:13:50.354666 on 04/09/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 216.114.205.133 | 6 | 6 | 3 | 3 |
| 63.166.4.59 | 5 | 5 | 1 | 1 |
| MY.NET.217.230 | 4 | 4 | 3 | 3 |
| 64.182.96.150 | 4 | 4 | 4 | 4 |
| MY.NET.213.42 | 2 | 2 | 1 | 1 |
| MY.NET.209.34 | 2 | 2 | 2 | 2 |
| MY.NET.203.150 | 1 | 1 | 1 | 1 |
| 63.202.222.239 | 1 | 1 | 1 | 1 |
| MY.NET.204.130 | 1 | 1 | 1 | 1 |
| 203.34.200.71 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 64.182.96.150 | 7 | 7 | 6 | 6 |
| MY.NET.229.130 | 5 | 6 | 1 | 2 |
| 216.114.205.133 | 5 | 5 | 4 | 4 |
| MY.NET.213.42 | 3 | 4 | 1 | 2 |
| 203.34.200.71 | 3 | 3 | 3 | 3 |
| MY.NET.217.134 | 2 | 2 | 1 | 1 |
| MY.NET.205.214 | 1 | 2 | 1 | 2 |
| 195.143.23.2 | 1 | 1 | 1 | 1 |
| 208.59.198.237 | 1 | 1 | 1 | 1 |
| MY.NET.70.242 | 1 | 1 | 1 | 1 |

This SNORT rule identifies traffic targeting and originating from port 65535/udp, which is usually associated with the Red Worm. Though much of this traffic is people looking for systems compromised by the Red Worm, of special note are the MY.NET hosts which are listed as sources above which should be checked to see if they have been compromised.

## Null scan (20 sources, 21 destinations)

61 alerts with this signature

Earliest such alert at 08:09:04.349571 on 04/03/2001
Latest such alert at 12:16:36.027607 on 04/09/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 192.12.78.2 | 31 | 31 | 1 | 1 |
| 24.17.64.12 | 4 | 4 | 1 | 1 |
| 24.229.55.174 | 3 | 3 | 1 | 1 |
| 164.76.175.213 | 3 | 3 | 1 | 1 |
| 217.3.182.110 | 3 | 57 | 3 | 41 |
| 24.28.13.149 | 2 | 2 | 1 | 1 |
| 193.11.231.49 | 2 | 2 | 1 | 1 |
| 65.8.10.224 | 1 | 1 | 1 | 1 |
| 24.25.240.218 | 1 | 1 | 1 | 1 |
| 24.6.97.144 | 1 | 1 | 1 | 1 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.219.38 | 31 | 151 | 1 | 12 |
| MY.NET.202.162 | 4 | 7 | 1 | 2 |
| MY.NET.208.166 | 3 | 3 | 1 | 1 |
| MY.NET.204.106 | 3 | 3 | 2 | 2 |
| MY.NET.221.110 | 3 | 3 | 1 | 1 |
| MY.NET.225.138 | 2 | 13 | 1 | 3 |
| MY.NET.206.22 | 1 | 2 | 1 | 2 |
| MY.NET.222.186 | 1 | 341 | 1 | 3 |
| MY.NET.222.90 | 1 | 1 | 1 | 1 |
| MY.NET.60.38 | 1 | 1 | 1 | 1 |

A NULL scan is where someone is attempting to map your network and/or services using TCP packets without any flags (SYN, FIN, RST. Etc.) set. This is an attempt to by-pass old/or improperly configured ID systems. Scanning in and of itself is not harmful to your systems, however, if they found what they were looking for you can expect a return visit. You can limit the amount of intelligence gathered by these types of scans by placing most of your systems behind a firewall and limiting the services that are running on your systems to the bare minimum.

For more information on this vulnerability see:
advICE#2000309, or aracnid#4

***NMAP TCP ping (8 sources, 46 destinations)***

63 alerts with this signature

Earliest such alert at 05:01:08.963859 on 04/03/2001
Latest such alert at 11:00:56.492476 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 217.3.182.110 | 50 | 57 | 39 | 41 |
| 194.133.58.2 | 3 | 3 | 3 | 3 |
| 200.52.109.160 | 3 | 3 | 1 | 1 |
| 194.133.58.129 | 3 | 3 | 2 | 2 |
| 204.155.48.3 | 1 | 1 | 1 | 1 |
| 202.187.24.3 | 1 | 1 | 1 | 1 |
| 213.8.52.189 | 1 | 1 | 1 | 1 |
| 199.197.130.21 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.1.8 | 3 | 6 | 1 | 3 |
| MY.NET.53.93 | 3 | 4 | 1 | 2 |
| MY.NET.53.220 | 2 | 2 | 1 | 1 |

| | | | | |
|---|---|---|---|---|
| MY.NET.53.114 | 2 | 3 | 1 | 1 |
| MY.NET.53.185 | 2 | 2 | 1 | 1 |
| MY.NET.53.175 | 2 | 3 | 1 | 2 |
| MY.NET.53.149 | 2 | 2 | 1 | 1 |
| MY.NET.53.228 | 2 | 2 | 1 | 1 |
| MY.NET.53.49 | 2 | 4 | 1 | 2 |
| MY.NET.1.3 | 2 | 2 | 2 | 2 |

This activity is fairly common on the Internet and involves the use of a very popular tool called Nmap. Someone is using the tool to map out your network, unfortunately because your systems don't sit behind a firewall the damage has already been done. The person running the tool now knows what systems (in the range they were scanning) are up, and what services they are running (in the port range that was scanned). This type of activity can be controlled if your systems reside behind a firewall.

## Queso fingerprint (19 sources, 30 destinations)

71 alerts with this signature

Earliest such alert at 03:30:56.437636 on 04/03/2001
Latest such alert at 23:06:21.031702 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 209.150.104.78 | 29 | 29 | 2 | 2 |
| 217.85.95.229 | 10 | 10 | 1 | 1 |
| 213.76.185.130 | 7 | 7 | 7 | 7 |
| 130.233.26.197 | 5 | 5 | 1 | 1 |
| 206.205.246.10 | 3 | 3 | 3 | 3 |
| 158.75.57.4 | 3 | 3 | 2 | 2 |
| 66.1.65.32 | 2 | 2 | 2 | 2 |
| 193.249.43.96 | 1 | 1 | 1 | 1 |
| 216.5.180.10 | 1 | 1 | 1 | 1 |
| 193.248.133.8 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.6.39 | 17 | 17 | 1 | 1 |
| MY.NET.6.44 | 12 | 15 | 1 | 2 |
| MY.NET.225.138 | 10 | 13 | 1 | 3 |
| MY.NET.219.134 | 5 | 5 | 1 | 1 |
| MY.NET.208.22 | 2 | 2 | 1 | 1 |
| MY.NET.217.214 | 1 | 1 | 1 | 1 |
| MY.NET.253.42 | 1 | 3 | 1 | 2 |
| MY.NET.253.43 | 1 | 14 | 1 | 5 |
| MY.NET.214.210 | 1 | 1 | 1 | 1 |

Queso is a popular tool for performing OS fingerprinting. By probing certain ports with certain types of packets and noting how your system responds, they can determine what operating systems you are running. With this information they can tailor future attacks to take advantage of OS-specific vulnerabilities. Make sure all of your systems are upgraded to the latest versions, with all applicable patches applied.

For more information on this vulnerability see:
CAN-1999-0454, advICE#2000313, or aracnid#29

## TCP SRC and DST outside network (24 sources, 49 destinations)

104 alerts with this signature

Earliest such alert at 00:43:16.664459 on 04/03/2001
Latest such alert at 23:33:10.340043 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 169.254.101.152 | 25 | 53 | 15 | 30 |
| 172.173.206.247 | 16 | 16 | 3 | 3 |
| 5.0.0.4 | 15 | 15 | 1 | 1 |
| 128.101.28.114 | 12 | 175 | 11 | 98 |
| 172.134.134.167 | 11 | 11 | 1 | 1 |
| 128.220.63.215 | 2 | 2 | 1 | 1 |
| 172.173.202.178 | 2 | 2 | 1 | 1 |
| 172.166.54.151 | 2 | 2 | 1 | 1 |
| 172.166.96.142 | 2 | 2 | 1 | 1 |
| 172.168.1.238 | 2 | 2 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 5.0.0.4 | 15 | 15 | 1 | 1 |
| 24.234.112.228 | 11 | 11 | 1 | 1 |
| 151.25.138.101 | 9 | 9 | 1 | 1 |
| 207.230.208.228 | 6 | 6 | 1 | 1 |
| 205.188.48.117 | 4 | 4 | 1 | 1 |
| 172.173.73.205 | 4 | 4 | 1 | 1 |
| 61.74.156.34 | 3 | 3 | 2 | 2 |
| 205.188.48.186 | 3 | 3 | 1 | 1 |
| 205.188.48.118 | 2 | 2 | 1 | 1 |
| 205.188.8.72 | 2 | 2 | 1 | 1 |

## Watchlist 000222 NET-NCFC (12 sources, 13 destinations)

106 alerts with this signature:

Earliest such alert at 05:35:21.884358 on 04/03/2001
Latest such alert at 21:00:33.035950 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 159.226.41.166 | 38 | 38 | 1 | 1 |
| 159.226.232.36 | 20 | 20 | 2 | 2 |
| 159.226.47.5 | 12 | 12 | 1 | 1 |
| 159.226.114.1 | 9 | 9 | 2 | 2 |
| 159.226.228.1 | 7 | 7 | 3 | 3 |
| 159.226.92.9 | 6 | 6 | 1 | 1 |
| 159.226.158.188 | 5 | 5 | 2 | 2 |
| 159.226.47.195 | 3 | 3 | 1 | 1 |
| 159.226.45.3 | 2 | 2 | 1 | 1 |
| 159.226.92.10 | 2 | 2 | 1 | 1 |
| 159.226.247.60 | 1 | 1 | 1 | 1 |
| 159.226.5.222 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.100.83 | 38 | 39 | 1 | 2 |
| MY.NET.4.3 | 12 | 12 | 1 | 1 |
| MY.NET.6.35 | 12 | 16 | 2 | 5 |

|  |  |  |  |  |
|---|---|---|---|---|
| MY.NET.6.34 | 11 | 14 | 1 | 3 |
| MY.NET.144.54 | 6 | 6 | 1 | 1 |
| MY.NET.253.43 | 6 | 14 | 3 | 5 |
| MY.NET.6.47 | 6 | 23 | 1 | 3 |
| MY.NET.6.7 | 5 | 6 | 2 | 3 |
| MY.NET.140.236 | 3 | 3 | 1 | 1 |
| MY.NET.145.9 | 2 | 2 | 1 | 1 |
| MY.NET.253.42 | 2 | 3 | 1 | 2 |
| MY.NET.253.41 | 2 | 19 | 1 | 3 |
| MY.NET.100.230 | 1 | 5 | 1 | 4 |

This SNORT rule identifies traffic targeting and/or originating from hosts on the 159.226 network (The Computer Network Center Chinese Academy of Sciences). These types of SNORT rules are created when networks and/or hosts become the source of repeated attacks and attempts to compromise systems. Any traffic to and from the 159.226 should immediately be suspect. Of special interest should be:

- telnet activity from 159.226.41.166 to MY.NET.100.83
- mail activity to MY.NET.6.34, MY.NET.6.35, and MY.NET.4.3

One way to prevent this type of activity would be to block the 159.226 network at your firewall.

## SMB Name Wildcard (99 sources, 90 destinations)

149 alerts with this signature

Earliest such alert at 01:53:47.991076 on 04/03/2001
Latest such alert at 19:03:16.579779 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 130.13.111.14 | 9 | 9 | 1 | 1 |
| 130.13.64.211 | 5 | 5 | 2 | 2 |
| MY.NET.111.156 | 4 | 5 | 1 | 2 |
| 61.74.165.26 | 3 | 3 | 1 | 1 |
| 169.254.195.73 | 3 | 3 | 1 | 1 |
| 213.64.113.73 | 3 | 3 | 1 | 1 |
| 130.13.147.94 | 3 | 3 | 1 | 1 |
| 61.180.155.99 | 3 | 3 | 1 | 1 |
| 24.0.157.82 | 2 | 2 | 1 | 1 |
| 130.101.12.217 | 2 | 2 | 2 | 2 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.135.135 | 10 | 12 | 2 | 4 |
| MY.NET.133.233 | 7 | 8 | 3 | 4 |
| MY.NET.132.93 | 6 | 7 | 3 | 4 |
| MY.NET.125.41 | 4 | 4 | 1 | 1 |
| MY.NET.134.116 | 3 | 6 | 1 | 4 |
| MY.NET.135.108 | 3 | 3 | 2 | 2 |
| MY.NET.134.235 | 3 | 7 | 2 | 4 |
| MY.NET.135.177 | 3 | 5 | 1 | 3 |
| MY.NET.133.33 | 3 | 3 | 1 | 1 |
| MY.NET.133.158 | 3 | 3 | 1 | 1 |

This alert is triggered by UDP traffic going from port 137 to port 137, and is associated with NetBIOS name lookups. This activity could be a result of misconfiguration (such as the traffic from MY.NET.111.156 to MY.NET.125.41) or an attempt by someone to determine the open shares available on your MS Windows machines. All MS Windows machines should be protected by a firewall that blocks incoming port 137-139 traffic.

## SUNRPC highport access (6 sources, 6 destinations)

282 alerts with this signature

Earliest such alert at 04:43:54.753092 on 04/03/2001
Latest such alert at 23:45:50.959116 on 04/08/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 216.7.148.244 | 216 | 216 | 1 | 1 |
| 64.12.163.199 | 34 | 34 | 1 | 1 |
| 209.85.37.71 | 29 | 30 | 1 | 1 |
| 64.81.28.146 | 1 | 1 | 1 | 1 |
| 205.188.6.89 | 1 | 1 | 1 | 1 |
| 209.10.41.242 | 1 | 1 | 1 | 1 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.221.2 | 216 | 216 | 1 | 1 |
| MY.NET.209.10 | 34 | 34 | 1 | 1 |
| MY.NET.218.50 | 29 | 30 | 1 | 1 |
| MY.NET.6.7 | 1 | 6 | 1 | 3 |
| MY.NET.145.190 | 1 | 1 | 1 | 1 |
| MY.NET.53.21 | 1 | 2 | 1 | 2 |

This SNORT rule is notorious for generating false positives. Whitehats.com (www.whitehats.com/info/IDS429) identified this alert as an attempt to get port information for RPC services. In truth, the activity that generated these alerts can be placed into one of two categories: 1) IRC activity, 87% of the activity involved IRC traffic to MY.NET.221.2 and MY.NET.218.50 (if these are not legitimate IRC servers, then they need to be checked to see if they have been compromised), 2) AOL Instant Messenger, the remainder of the traffic originated from 64.12.163.199 (AOL) on port 9898 and went to MY.NET.209.10. It would be a good idea to review the use of AOL Instant Messenger at your site, as there are numerous security problems associated with its use.

## UDP SRC and DST outside network (46 sources, 274 destinations)

504 alerts with this signature

Earliest such alert at 00:00:04.563957 on 04/03/2001
Latest such alert at 21:38:43.746545 on 04/09/2001

| Top Ten Source | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 128.101.28.114 | 163 | 175 | 88 | 98 |
| 169.254.67.123 | 140 | 140 | 131 | 131 |
| 169.254.101.152 | 26 | 53 | 13 | 30 |
| 192.168.0.53 | 22 | 22 | 1 | 1 |

| | | | | |
|---|---|---|---|---|
| 204.62.41.254 | 16 | 16 | 1 | 1 |
| 169.254.114.199 | 12 | 12 | 5 | 5 |
| 192.168.0.2 | 11 | 13 | 2 | 4 |
| 200.200.200.13 | 9 | 9 | 1 | 1 |
| 169.254.26.24 | 8 | 8 | 6 | 6 |
| 169.254.236.29 | 7 | 7 | 1 | 1 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 208.48.72.124 | 37 | 37 | 3 | 3 |
| 164.124.101.2 | 26 | 26 | 7 | 7 |
| 10.10.10.50 | 22 | 22 | 1 | 1 |
| 169.254.15.217 | 21 | 22 | 1 | 1 |
| 204.62.32.194 | 16 | 16 | 1 | 1 |
| 192.168.0.1 | 14 | 14 | 9 | 9 |
| 207.66.83.101 | 10 | 10 | 1 | 1 |
| 209.87.79.232 | 9 | 9 | 1 | 1 |
| 198.6.1.2 | 8 | 8 | 2 | 2 |
| 24.3.0.34 | 6 | 6 | 1 | 1 |

This SNORT rule alerts on any UDP traffic that is from a system outside your network to a system outside your network. You should not see traffic like this, and most of it is probably due to improperly configured network devices or systems (for instance, the 10.10.10.50 and 192.168.XXX.XXX traffic is using reserved IP addresses and should never be seen). Another interesting item is that all of this traffic is port 137 to port 137, which would lead me to conclude that you have some improperly configured MS Windows machines.

## External RPC call (15 sources, 411 destinations)

513 alerts with this signature

Earliest such alert at 00:11:29.574031 on 04/03/2001
Latest such alert at 16:39:02.803253 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 216.104.105.66 | 84 | 84 | 78 | 78 |
| 152.20.21.60 | 75 | 75 | 72 | 72 |
| 203.69.227.45 | 75 | 75 | 75 | 75 |
| 195.24.196.199 | 60 | 60 | 60 | 60 |
| 24.43.176.96 | 56 | 56 | 56 | 56 |
| 196.36.119.123 | 29 | 29 | 29 | 29 |
| 128.148.184.75 | 28 | 28 | 28 | 28 |
| 200.214.212.2 | 28 | 28 | 27 | 27 |
| 210.99.13.253 | 21 | 21 | 21 | 21 |
| 198.135.204.114 | 17 | 17 | 17 | 17 |
| 24.232.100.216 | 14 | 14 | 14 | 14 |
| 211.34.177.194 | 9 | 9 | 9 | 9 |
| 210.255.74.250 | 7 | 7 | 7 | 7 |
| 64.245.9.146 | 7 | 7 | 7 | 7 |
| 151.39.246.114 | 3 | 3 | 3 | 3 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.133.232 | 4 | 6 | 4 | 6 |
| MY.NET.134.225 | 3 | 4 | 3 | 4 |
| MY.NET.134.127 | 3 | 3 | 2 | 2 |
| MY.NET.134.223 | 3 | 3 | 3 | 3 |
| MY.NET.133.180 | 3 | 4 | 3 | 4 |
| MY.NET.134.163 | 3 | 3 | 3 | 3 |
| MY.NET.134.146 | 3 | 5 | 2 | 3 |
| MY.NET.134.147 | 3 | 3 | 2 | 2 |

|  |  |  |  |  |
|---|---|---|---|---|
| MY.NET.132.38 | 3 | 5 | 3 | 5 |
| MY.NET.132.53 | 3 | 5 | 3 | 5 |

This SNORT rule identifies traffic targeting port 111, which is almost always associated with UNIX RPC portmappers. Due to the large number of vulnerabilities associated with this port, makes it one of the top five ports on the Internet scanned. The only way to stop this type of scanning is to place all of your machines behind a firewall and block port 111 at the firewall. If you don't require the portmapper to be running, turn it off. Otherwise, insure that you have all the necessary patches installed.

## Connect to 515 from outside (19 sources, 549 destinations)

819 alerts with this signature

Earliest such alert at 00:01:22.985557 on 04/03/2001
Latest such alert at 02:38:00.326667 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 140.122.140.57 | 165 | 165 | 165 | 165 |
| 148.202.15.214 | 115 | 115 | 115 | 115 |
| 207.8.203.106 | 106 | 106 | 106 | 106 |
| 216.130.139.13 | 87 | 87 | 83 | 83 |
| 207.102.158.10 | 67 | 67 | 57 | 57 |
| 64.18.0.162 | 55 | 55 | 55 | 55 |
| 64.14.243.59 | 44 | 44 | 43 | 43 |
| 24.27.245.64 | 26 | 26 | 24 | 24 |
| 200.10.244.200 | 24 | 24 | 24 | 24 |
| 24.219.83.24 | 19 | 19 | 17 | 17 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.133.12 | 5 | 5 | 5 | 5 |
| MY.NET.133.177 | 4 | 4 | 3 | 3 |
| MY.NET.133.104 | 4 | 4 | 4 | 4 |
| MY.NET.133.80 | 4 | 5 | 4 | 5 |
| MY.NET.133.102 | 4 | 4 | 4 | 4 |
| MY.NET.134.122 | 4 | 4 | 3 | 3 |
| MY.NET.133.126 | 4 | 4 | 4 | 4 |
| MY.NET.134.130 | 4 | 5 | 3 | 4 |
| MY.NET.135.130 | 4 | 6 | 4 | 6 |
| MY.NET.132.56 | 4 | 5 | 4 | 5 |

This SNORT rule identifies traffic that is originating from outside your network and targeting port 515 (UNIX/LINUX Print Services). Unless you are allowing people from the outside to print on your print servers (which is not a good idea), this traffic is almost certainly scanning for LINUX boxes running LPRng. Though there were many who took advantage of the vulnerabilities associated with port 515, it was the Ramen Worm (and subsequent worms) that brought port 515 scanning to new heights. If possible, you should disable print services on all but the recognized print servers, and insure that you have the lastest patches on those. Another safety precaution would be to place all of your systems behind a firewall, and block port 515.

## WinGate 1080 Attempt (79 sources, 2441 destinations)

2802 alerts with this signature

Earliest such alert at 00:55:52.539118 on 04/03/2001
Latest such alert at 22:45:41.832393 on 04/09/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 24.29.190.75 | 285 | 561 | 257 | 454 |
| 168.122.242.151 | 165 | 322 | 152 | 265 |
| 128.104.136.68 | 162 | 317 | 154 | 264 |
| 128.211.227.112 | 159 | 159 | 145 | 145 |
| 128.104.53.41 | 143 | 300 | 128 | 224 |
| 209.124.86.84 | 139 | 257 | 117 | 189 |
| 149.159.46.132 | 135 | 248 | 111 | 186 |
| 216.161.84.223 | 132 | 246 | 121 | 207 |
| 24.200.164.97 | 128 | 247 | 112 | 204 |
| 24.112.236.61 | 123 | 244 | 108 | 182 |

| Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.60.11 | 9 | 10 | 4 | 5 |
| MY.NET.98.141 | 7 | 7 | 3 | 3 |
| MY.NET.204.22 | 7 | 8 | 3 | 4 |
| MY.NET.60.8 | 5 | 6 | 5 | 6 |
| MY.NET.211.250 | 5 | 6 | 2 | 3 |
| MY.NET.202.58 | 4 | 4 | 1 | 1 |
| MY.NET.98.115 | 4 | 8 | 2 | 2 |
| MY.NET.10.121 | 3 | 3 | 1 | 1 |
| MY.NET.219.46 | 3 | 4 | 2 | 3 |
| MY.NET.231.216 | 3 | 4 | 1 | 1 |

This SNORT rule identifies traffic targeting port 1080, which is used by an MS Windows-based proxy software called Wingate. If they find an unprotected proxy server they may use it to attack another system and make it look like the attack came from your server. Proxy servers should always be protected by a firewall and/or access control list.

For more information on this vulnerability see:
bugtraq#154, or aracnid#481

## SYN-FIN scan (2 sources, 2693 destinations)

2849 alerts with this signature

Earliest such alert at 14:41:12.382458 on 04/03/2001
Latest such alert at 11:40:21.763870 on 04/05/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 210.96.75.129 | 1447 | 1447 | 1447 | 1447 |
| 211.178.63.4 | 1402 | 1402 | 1305 | 1305 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.15.246 | 3 | 3 | 2 | 2 |
| MY.NET.12.183 | 3 | 3 | 2 | 2 |
| MY.NET.223.219 | 3 | 3 | 2 | 2 |

|  |  |  |  |  |
|---|---|---|---|---|
| MY.NET.199.241 | 3 | 3 | 2 | 2 |
| MY.NET.167.166 | 3 | 3 | 2 | 2 |
| MY.NET.205.171 | 3 | 5 | 2 | 3 |
| MY.NET.134.235 | 3 | 7 | 1 | 4 |
| MY.NET.146.195 | 3 | 3 | 1 | 1 |
| MY.NET.171.164 | 2 | 2 | 1 | 1 |
| MY.NET.210.194 | 2 | 3 | 2 | 3 |

SYN-FIN scans, as the name suggests, use specially crafted TCP packets with the SYN and FIN flags set, to map networks. The combination of SYN and FIN flags will never occur naturally and are able to penetrate undetected older ID systems. The use of crafted packets is confirmed by corresponding entries in the OOS (Out Of Spec) files. This type of scanning will reveal open ports on your systems, and is usually a precursor to attacks against vulnerable services. The only real defense is to keep the number of services running to a minimum, keep your systems patched, and place your systems behind a firewall.

For more information on this vulnerability see:
aracnid#198

## Possible RAMEN server activity (820 sources, 3439 destinations)

4994 alerts with this signature

Earliest such alert at 00:16:00.894397 on 04/03/2001
Latest such alert at 12:58:49.518110 on 04/03/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.15.214 | 1596 | 10241 | 1178 | 7962 |
| 24.29.190.75 | 276 | 561 | 257 | 454 |
| 128.104.53.41 | 157 | 300 | 139 | 224 |
| 168.122.242.151 | 157 | 322 | 145 | 265 |
| 128.104.136.68 | 155 | 317 | 140 | 264 |
| 24.112.236.61 | 121 | 244 | 106 | 182 |
| 24.200.164.97 | 119 | 247 | 112 | 204 |
| 209.124.86.84 | 118 | 257 | 108 | 189 |
| 216.161.84.223 | 114 | 246 | 110 | 207 |
| 149.159.46.132 | 113 | 248 | 101 | 186 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 128.104.136.68 | 162 | 162 | 141 | 141 |
| MY.NET.15.214 | 152 | 1550 | 127 | 1298 |
| 168.122.242.151 | 150 | 150 | 137 | 137 |
| 128.255.166.54 | 80 | 80 | 70 | 70 |
| 216.78.180.74 | 60 | 60 | 51 | 51 |
| 24.29.190.75 | 39 | 39 | 35 | 35 |
| 24.112.236.61 | 38 | 38 | 31 | 31 |
| 128.206.234.79 | 37 | 37 | 29 | 29 |
| 204.210.131.190 | 23 | 23 | 19 | 19 |
| 209.124.86.84 | 20 | 20 | 16 | 16 |

This SNORT rule identifies traffic that either targets or originates from port 27374, which is associated with the RAMEN worm. Of immediate interest is the traffic involving MY.NET.15.214, due to the high level of activity to and from this box it is safe to say that it has been compromised. Remove it immediately from service and reload it. The remaining activity appears to be people scanning for RAMEN compromised systems.

For more information on this vulnerability see:
aracnid#460

## Attempted Sun RPC high port access (1 source, 1 destination)

5177 alerts with this signature

Earliest such alert at 19:55:24.753839 on 04/09/2001
Latest such alert at 22:17:27.809503 on 04/09/2001

| Source | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 66.26.3.204 | 5177 | 5177 | 1 | 1 |

| Destination | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.217.242 | 5177 | 5177 | 1 | 1 |

This SNORT rule identifies traffic targeting port 32771, which is associated with Sun RPC. Due to the volume of traffic hitting MY.NET.217.242 from a single host, it is safe to say that this box has been compromised. Remove it immediately from service and reload it. To prevent people from accessing port 32771, you should place your systems behind a firewall and block all access to this port.

## High port 65535 tcp possible Red Worm – traffic (17 sources, 5459 destinations)

6973 alerts with this signature

Earliest such alert at 21:40:11.368859 on 04/03/2001
Latest such alert at 21:00:37.956214 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.253.12 | 6922 | 6922 | 5441 | 5441 |
| MY.NET.221.90 | 9 | 9 | 1 | 1 |
| MY.NET.99.51 | 6 | 6 | 1 | 1 |
| 129.59.51.185 | 6 | 6 | 1 | 1 |
| MY.NET.6.44 | 6 | 6 | 1 | 1 |
| MY.NET.253.51 | 5 | 9 | 1 | 2 |
| MY.NET.6.35 | 3 | 7 | 3 | 5 |
| 64.50.191.56 | 3 | 3 | 1 | 1 |
| 140.113.146.60 | 3 | 3 | 3 | 3 |
| 64.37.200.46 | 2 | 2 | 1 | 1 |
| MY.NET.204.66 | 2 | 2 | 1 | 1 |
| MY.NET.253.41 | 1 | 1 | 1 | 1 |
| MY.NET.253.24 | 1 | 6 | 1 | 4 |
| 146.145.176.8 | 1 | 1 | 1 | 1 |
| MY.NET.178.42 | 1 | 22201 | 1 | 4 |
| MY.NET.204.18 | 1 | 2 | 1 | 2 |
| 154.11.89.182 | 1 | 1 | 1 | 1 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| 129.59.51.185 | 12 | 12 | 3 | 3 |
| MY.NET.122.249 | 6 | 6 | 1 | 1 |
| 64.50.191.56 | 6 | 6 | 1 | 1 |
| MY.NET.204.66 | 6 | 6 | 1 | 1 |
| 128.227.244.128 | 6 | 6 | 1 | 1 |
| 205.188.157.25 | 5 | 5 | 1 | 1 |
| MY.NET.62.239 | 5 | 5 | 1 | 1 |
| MY.NET.56.169 | 5 | 5 | 1 | 1 |
| MY.NET.44.14 | 4 | 4 | 1 | 1 |
| MY.NET.239.109 | 4 | 4 | 1 | 1 |

This SNORT rule identifies traffic targeting and originating from port 65535/tcp, which is usually associated with the Red Worm. Though much of this traffic is people looking for systems compromised by the Red Worm, of special note are the MY.NET hosts which are listed as sources above (especially MY.NET.253.12) which should be checked to see if they have been compromised.

## Watchlist 000220 IL-ISDNNET-990517 (41 sources, 36 destinations)

10144 alerts with this signature

Earliest such alert at 03:07:19.770005 on 04/03/2001
Latest such alert at 21:36:04.414291 on 04/09/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| 212.179.5.84 | 3600 | 3600 | 1 | 1 |
| 212.179.79.2 | 2206 | 2206 | 5 | 5 |
| 212.179.80.79 | 677 | 677 | 1 | 1 |
| 212.179.84.195 | 614 | 614 | 1 | 1 |
| 212.179.80.232 | 518 | 518 | 1 | 1 |
| 212.179.80.225 | 417 | 417 | 1 | 1 |
| 212.179.24.155 | 396 | 396 | 2 | 2 |
| 212.179.5.90 | 334 | 334 | 1 | 1 |
| 212.179.125.114 | 291 | 291 | 2 | 2 |
| 212.179.82.254 | 216 | 216 | 1 | 1 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.218.142 | 3600 | 3600 | 1 | 1 |
| MY.NET.204.122 | 1659 | 1659 | 3 | 3 |
| MY.NET.205.246 | 835 | 837 | 1 | 3 |
| MY.NET.205.6 | 677 | 678 | 1 | 2 |
| MY.NET.221.14 | 614 | 614 | 1 | 1 |
| MY.NET.208.34 | 518 | 519 | 1 | 2 |
| MY.NET.214.50 | 417 | 417 | 1 | 1 |
| MY.NET.98.159 | 395 | 395 | 1 | 1 |
| MY.NET.222.186 | 340 | 341 | 2 | 3 |
| MY.NET.178.42 | 294 | 4108 | 2 | 5 |

This SNORT rule identifies traffic targeting and/or originating from hosts on the 212.179 network (The ISDN Network in Israel). These types of SNORT rules are created when networks and/or hosts become the source of repeated attacks and attempts to compromise systems. Any traffic to and from the 212.179 should immediately be suspect. Hosts targeted from this network should be checked for signs of compromise.

One way to prevent this type of activity would be to block the 212.179 network at your firewall.

## Possible trojan server activity (1327 sources, 7814 destinations)

11280 alerts with this signature

Earliest such alert at 20:20:26.054216 on 04/03/2001
Latest such alert at 23:19:35.820285 on 04/09/2001

| Top Ten Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.15.214 | 8645 | 10241 | 6791 | 7962 |
| 24.112.202.176 | 913 | 913 | 858 | 858 |
| MY.NET.98.193 | 59 | 59 | 48 | 48 |
| 206.132.75.244 | 13 | 13 | 1 | 1 |
| 24.188.217.161 | 8 | 8 | 8 | 8 |
| 211.219.138.228 | 7 | 11 | 5 | 8 |
| MY.NET.219.86 | 6 | 6 | 3 | 3 |
| 211.135.37.98 | 6 | 6 | 6 | 6 |
| 24.180.160.210 | 6 | 6 | 3 | 3 |
| 130.205.77.148 | 5 | 5 | 3 | 3 |

| Top Ten Destinations | # Alerts (sig) | # Alerts (total) | # Srcs (sig) | # Srcs (total) |
|---|---|---|---|---|
| MY.NET.15.214 | 1398 | 1550 | 1171 | 1298 |
| 24.112.202.176 | 91 | 91 | 65 | 65 |
| MY.NET.253.41 | 13 | 19 | 1 | 3 |
| MY.NET.219.86 | 6 | 7 | 4 | 5 |
| 200.64.239.65 | 4 | 4 | 1 | 1 |
| MY.NET.206.38 | 4 | 5 | 3 | 4 |
| 24.132.57.11 | 4 | 4 | 1 | 1 |
| MY.NET.181.171 | 4 | 4 | 1 | 1 |
| 208.25.153.26 | 4 | 4 | 1 | 1 |
| 200.64.237.55 | 4 | 4 | 1 | 1 |

This SNORT rule identifies traffic targeting and originating from port 27374, which is associated with SubSeven version 2. Though much of this traffic is people looking for SubSeven compromised hosts, of special note are the MY.NET hosts that are listed as sources above (especially MY.NET.15.214). These machines should be checked for signs of having been compromised.

## Russia Dynamo - SANS Flash 28-jul-00 (4 sources, 4 destinations)

26014 alerts with this signature

Earliest such alert at 18:20:18.157683 on 04/04/2001
Latest such alert at 23:49:57.698982 on 04/09/2001

| Sources | # Alerts (sig) | # Alerts (total) | # Dsts (sig) | # Dsts (total) |
|---|---|---|---|---|
| MY.NET.178.42 | 22200 | 22201 | 3 | 4 |

| | | | | |
|---|---|---|---|---|
| 194.87.6.106 | 2763 | 2763 | 1 | 1 |
| 194.87.6.33 | 536 | 536 | 1 | 1 |
| 194.87.6.21 | 515 | 515 | 1 | 1 |

| **Destinations** | **# Alerts (sig)** | **# Alerts (total)** | **# Srcs (sig)** | **# Srcs (total)** |
|---|---|---|---|---|
| 194.87.6.106 | 19501 | 19501 | 1 | 1 |
| MY.NET.178.42 | 3814 | 4108 | 3 | 5 |
| 194.87.6.21 | 1599 | 1599 | 1 | 1 |
| 194.87.6.33 | 1100 | 1100 | 1 | 1 |

This is a special SNORT rule that is associated with unusual activity involving dol.ru (194.87 or 194.87.6), first reported on 29 July 2000 by SANS incident handler Stephen Northcutt (http://www.sans.org/y2k/072818.htm). According to his report, you should immediately remove MY.NET.178.42 from service and reload it.

## *SCAN DATA*

For the period examined, there were 194429 entries in the SNORT Scan Reports. These entries are broken down as follows:

**113585 entries UDP**

UDP scans are very common and in of themselves, not cause for concern. What should be of concern are the fact that all of the Top Ten Sources of the scans are your boxes. You should check these boxes for signs of compromise.

| Top Ten Sources | # of Hits |
|---|---|
| MY.NET.217.242 | 7755 |
| MY.NET.202.34 | 7154 |
| MY.NET.217.230 | 5246 |
| MY.NET.226.190 | 3814 |
| MY.NET.217.134 | 3651 |
| MY.NET.227.222 | 3442 |
| MY.NET.98.162 | 3253 |
| MY.NET.209.42 | 3070 |
| MY.NET.222.54 | 2785 |
| MY.NET.205.214 | 2849 |

| Top Ten Destinations | # of Hits |
|---|---|
| MY.NET.218.26 | 1280 |
| 209.150.227.138 | 1150 |
| 205.229.210.44 | 1145 |
| 66.26.3.204 | 1118 |
| 63.29.237.141 | 1032 |
| 63.121.232.185 | 917 |
| 208.191.190.4 | 749 |
| 24.180.11.253 | 619 |
| 24.21.203.64 | 469 |
| 142.166.220.84 | 456 |

9275 of the UDP Scan entries appear to be Starsiege Tribes activity (http://www.sierra.com/support/technical/product documents/tribests.html) involving the following MY.NET hosts:
MY.NET.160.138
MY.NET.220.190
MY.NET.150.145
MY.NET.97.158
MY.NET.229.50
MY.NET.228.54
MY.NET.98.162
MY.NET.98.205
MY.NET.226.190
MY.NET.222.134
MY.NET.97.166
MY.NET.229.130
These systems should be checked, if they are MS Windows machines look for the Tribes game.

**77845 entries SYN**

SYN scans are the most common form of scanning. They can be used to find hosts that are up, and/or ports that are open. The only thing to be concerned about is if you find that your hosts are initiating the scans. As you can see, MY.NET.204.18 and MY.NET.15.214 are both scanning. If you are not using them to conduct security scans of your own network, then it is a good bet both systems have been compromised.

| **Top Ten Sources** | **# of hits** |
|---|---|
| MY.NET.204.18 | 21440 |
| MY.NET.15.214 | 17053 |
| 64.229.232.100 | 3864 |
| 202.145.57.82 | 3231 |
| 210.111.248.98 | 2880 |
| 165.246.154.57 | 2810 |
| 144.230.171.194 | 2455 |
| 211.57.209.226 | 2356 |
| 200.24.214.131 | 2319 |
| 216.156.140.50 | 1686 |

| **Top Ten Destinations** | **# of hits** |
|---|---|
| 63.88.120.21 | 21162 |
| 24.13.123.8 | 619 |
| MY.NET.162.64 | 315 |
| 129.21.112.10 | 278 |
| MY.NET.162.75 | 236 |
| 131.183.38.37 | 44 |
| 79.72.97.40 | 44 |
| 205.188.253.228 | 41 |
| 66.191.158.62 | 40 |
| 195.134.34.162 | 39 |

## 2501 entries SYN-FIN

These SYN-FIN scans correspond to the ones reported by the SNORT alerts and are explained there.

| **Sources** | **# of hits** |
|---|---|
| 210.96.75.129 | 1443 |
| 211.178.63.4 | 1055 |
| MY.NET.227.130 | 1 |
| MY.NET.225.42 | 1 |
| 24.66.225.57 | 1 |

| **Top Ten Destinations** | **# of hits** |
|---|---|
| MY.NET.134.235 | 3 |
| MY.NET.146.195 | 3 |
| MY.NET.15.246 | 3 |
| MY.NET.199.241 | 3 |
| MY.NET.223.219 | 3 |
| MY.NET.105.246 | 2 |
| MY.NET.105.209 | 2 |
| MY.NET.105.195 | 2 |
| MY.NET.104.246 | 2 |
| MY.NET.102.247 | 2 |

## 300 entries NULL

These NULL scans correspond to the ones reported by the SNORT alerts, more information can be found there.

| **Top Ten Sources** | **# of hits** |
|---|---|
| MY.NET.206.146 | 215 |
| 192.12.78.2 | 31 |
| MY.NET.227.130 | 16 |
| 24.17.64.12 | 4 |
| MY.NET.225.42 | 4 |

| | |
|---|---|
| 164.76.175.213 | 3 |
| 24.229.55.174 | 3 |
| 24.28.13.149 | 2 |
| MY.NET.223.74 | 2 |
| 134.96.56.232 | 1 |

| **Top Ten Destinations** | **# of hits** |
|---|---|
| 207.226.225.3 | 215 |
| MY.NET.219.38 | 31 |
| MY.NET.202.162 | 4 |
| 170.140.23.35 | 3 |
| 65.24.213.207 | 3 |
| MY.NET.208.166 | 3 |
| MY.NET.221.110 | 3 |
| MY.NET.225.138 | 3 |
| 24.180.132.123 | 2 |
| 217.1.123.176 | 2 |

## 198 entries Everything Remaining

What remains is a collection of unusual activity that is difficult to categorize. It is probably wide level scanning of class B networks, and yours happen to be hit. Due to the low volume, I would not be too concerned.

| **Top Ten Sources** | **# of hits** |
|---|---|
| MY.NET.227.130 | 17 |
| 213.98.52.250 | 10 |
| MY.NET.225.42 | 10 |
| 164.76.175.213 | 7 |
| 207.164.170.84 | 7 |
| 24.8.137.214 | 6 |
| 24.108.107.170 | 5 |
| 205.151.64.161 | 5 |
| MY.NET.205.54 | 4 |
| 24.169.42.154 | 4 |

| **Top Ten Destinations** | **# of hits** |
|---|---|
| MY.NET.222.230 | 11 |
| MY.NET.221.14 | 7 |
| MY.NET.221.110 | 7 |
| MY.NET.208.166 | 7 |
| MY.NET.205.142 | 6 |
| MY.NET.204.98 | 5 |
| MY.NET.225.138 | 4 |
| MY.NET.219.38 | 4 |
| MY.NET.210.46 | 4 |
| MY.NET.202.162 | 4 |

## *Out of Spec (OOS) Data*

Out-of-Spec packets are those that SNORT has detected that should not, according to the RFCs , appear naturally. There were, for the time period covered by this analysis, a total of 3846 out-of-spec packets detected. Keep in mind that packets can and do become corrupted everyday, having some packets out-of-spec is normal. Though I could not account for each and every one, I was able to find the source of most of them:

**3203 packets SYN-FIN Scan**
These packets correspond to the SYN-FIN scans detected by the SNORT alerts. The reason they also appear on this report is that the existence of SYN and FIN flags set on the same packet is unusual, and normally a sign of packet crafting.

**70 packets Queso OS Fingerprinting**
These packets correspond to the Queso fingerprinting detected by the SNORT alerts. The reason they also appear here is that in order to determine that operating system of a target host, Queso crafts an illegal packet to see how it responds. This packet has a Time To Live (TTL) value greater than 225 and has two reserve flags set.

**18 packets NMAP OS Fingerprinting**
These packets correspond to the NMAP fingerprinting detected by the SNORT alerts. The reason they appear here is because NMAP sends various types of abnormal packets to determine the operating system of a remote host. The packets found here, that also match the SNORT alert, have the URG,PSH, SYN, and FIN flags set.

## *SUMMARY*

**TOP TEN SOURCES**
MY.NET.178.42
MY.NET.15.214
MY.NET.253.12
66.26.3.204
212.179.5.84
194.87.6.106
212.179.79.2
210.96.75.129
211.178.63.4
24.112.202.176

The fact that the top three systems on the sources list belong to you should be of concern. MY.NET.178.42 was the primary source of the Russia Dynamo activity. MY.NET.15.214 was the primary source of Trojan server activity as well as port scanning. MY.NET.253.12 was the primary source of possible Red Worm traffic. These three system should be checked for signs of compromise.

**TOP TEN DESTINATIONS**
194.87.6.106
MY.NET.217.242
MY.NET.178.42
MY.NET.218.142
MY.NET.204.122
194.87.6.21
MY.NET.15.214
194.87.6.33
MY.NET.205.246
MY.NET.205.6

All of the MY.NET systems on the destinations list should be checked for signs of compromise. Even if it was just targeted by a scanner, the next step would be to exploit a vulnerability related to the port or OS discovered.

## *DEFENSIVE RECOMMENDATIONS*

Most of my defensive recommendations are going to be common sense:

- Considering the amount of gaming we found, a computer usage policy needs to be developed and enforced.
- Configuration management is the key to keeping vulnerabilities mentioned on Bugtraq from becoming your personal problem. All workstations and servers that perform a similar function should be built from a carefully control build tape. A database, or at least a spreadsheet, should be maintained that outlines what software and version are load on each. That way when a vulnerability or patch is announced, it is easy to identify what boxes need to be upgraded.
- Turn off all unnecessary services. Many OS load all available services by default. This not only wastes processing cycles and disk space, but it also opens you up to all sort of attacks. If your workstations or servers don't have a printer connected to them, remove LPR, etc.
- Prepare a layered defense. Install firewalls and filters at the network and enclave levels to limit access to your systems, and install virus checkers and personal firewalls on your servers and workstations. That way is someone were to get past your outer layer of defenses, you are not totally open to attack. Also employ egress filtering on your firewalls to prevent your systems from being used to attack other systems.
- Look for more secure versions of network applications, for example, instead of using telnet, use ssh.

## *DATA ANALYSIS*

I used a number of scripts, applications, and processes to analyze the ~26 MB of data. First I prepared the alert data for analysis, by searching for every occurrence of MY.NET and changing it to 10.1. To do this I used a simple Bourne Shell script:

```
#!/bin/sh
```

```
for infile in `/bin/ls -1 alert*`
do
cat $infile | sed 's/MY.NET/10.1/g' > m$infile
done
```

I then used another Bourne Shell script to combine all of the alert files into a single file:

```
#!/bin/sh
for infile in `/bin/ls -1 malert.*`
do
cat $infile >> master.alert
done
```

Next I used SnortSnarf v.052301.1 to summarize the alerts:

```
./snortsnarf.pl –d . master.alert &
```

I then used the following sources to research the alerts reported:

www.incidents.org
www.whitehats.com
www.cert.org
packetstorm.securify.com

Next I examined the scan and oos data. Unfortunately, the sources of the alerts and the sources of the scans/oos appeared to be different machines, because the time was different for the events reports. Correlation of the data was done by dropping the seconds off the time and comparing the ips and ports. This approach wasn't perfect, but seemed to get the job done.
The Top Ten Sources data was obtained by using grep to pull the related scan data out of the scan files and placing it in a separate file. I then ran the following PERL script on the data, while passing it through the UNIX sort command:

```
#!/usr/local/bin/perl
# count_sources.pl
%src = ();
while(<>) {
  if (/(\w+\.\w+\.\w+\.\w+)\:?\w{0,5} -> \w+\.\w+\.\w+\.\w+\:?\w{0,5}/) {
    if ( exists $src{$1} ) {
      $src{$1} += 1;
    } else {
      $src{$1} = 1;
    }
  }
}
foreach $sip (sort keys %src) {
  print "$sip $src{$sip}\n";
}
```

```
./count_sources.pl <SCAN>.dat | sort –r –k 2,2n
```

I used a similar approach for the Top Ten Destination data:

```
#!/usr/local/bin/perl
# count_destinations.pl
%dst = ();
while(<>) {
  if (/\w+\.\w+\.\w+\.\w+\:?\w{0,5} -> (\w+\.\w+\.\w+\.\w+)\:?\w{0,5}/) {
    if ( exists $dst{$1} ) {
      $dst{$1} += 1;
    } else {
      $dst{$1} = 1;
    }
  }
}
foreach $dip (sort keys %dst) {
  print "$dip $dst{$dip}\n";
}

./count_destinations.pl <SCAN>.dat | sort –r –k 2,2n
```

This data was cut and pasted into this document.

## *Acknowledgements*

I would like to acknowledge the following sources of information and ideas for performing this practical:

Varine, Brian. “SANS GIAC Certification, GCIA Practical Assignment V 2.7.” 28 January 2001. URL: http://www.sans.org/y2k/practical/Brian_Varine_GCIA.doc (24 June 2001).

Oborn, David. “SANS GCIA Practical Assignment.”
URL: http://www.sans.org/y2k/practical/David_Oborn_GCIA.html (24 June 2001)

Singer, David. “GIAC Practical”
URL: http://www.sans.org/y2k/practical/David_Singer_GCIA.doc (24 June 2001)

Asadoorian, Paul. “Intrusion Detection in Depth GCIA Practical Assignment V 2.8”
URL: http://www.sans.org/y2k/practical/Paul_Asadoorian_GIAC.doc (24 June 2001)

Bayerkohler, Marc. “SANS Intrusion Detection Practical”
URL: http://www.sans.org/y2k/practical/Marc_Bayerkohler_GIAC.html (24 June 2001)

Bruneau, Guy. “SANS GIAC Intrusion Detection Curriculum”
URL: http://www.sans.org/y2k/practical/Guy_Bruneau.doc (24 June 2001)

Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Reading: Addison Wesley Longman, Inc, 1994.

Northcutt, Stephen and Novak, Judy. Network Intrusion Detection: An Analyst Handbook, 2 ed. New Riders Publishing. 2001.