GIAC Certified Intrusion Analyst (GCIA)

Practical Assignment

Version 3.4

Paul Schmelzel
Submitted: April 27, 2004

Table of Contents

Abstract

This paper consists of three parts. The first part takes a look into the decoy scan option of Nmap. I had been unable to find papers that focus on this option and I had an interest in it to see what capabilities existed and if any trends could be discovered. The second part of the paper focuses on three real-world detects. These are not created alerts and two of them were taken from a real business environment. The final part of the paper analyzes five days worth of logs provided by www.incidents.org. The logs were analyzed and a full report is given.

Part 1

IDS Challenge - Nmap Decoy Scans

Nmap is one of the most well-known and most utilized security tools in the security industry today. If a port scan is necessary, Nmap is almost always the tool that will be called upon to perform it. It has even become part of the default build in many Linux distributions. Nmap can be found at http://www.insecure.org. Here is an excerpt from the homepage:

“Nmap ("Network Mapper") is an open source utility for network exploration or security auditing. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (application name and version) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics. Nmap runs on most types of computers, including Linux/BSD/Mac OS X, and Windows. Both console and graphical versions are available.”

Nmap includes a variety of options for scanning purposes to discover open ports. Below is the output from Nmap command-line help.

```
Nmap 3.50 Usage: nmap [Scan Type(s)] [Options] <host or net list>
Some Common Scan Types ('*' options require root privileges)
* -sS TCP SYN stealth port scan (default if privileged (root))
  -sT TCP connect() port scan (default for unprivileged users)
* -sU UDP port scan
  -sP ping scan (Find any reachable machines)
* -sF,-sX,-sN Stealth FIN, Xmas, or Null scan (experts only)
  -sV Version scan probes open ports determining service & app names/versions
  -sR/-I RPC/Identd scan (use with other scan types)
Some Common Options (none are required, most can be combined):
* -O Use TCP/IP fingerprinting to guess remote operating system
  -p <range> ports to scan.  Example range: '1-1024,1080,6666,31337'
  -F Only scans ports listed in nmap-services
  -v Verbose. Its use is recommended.  Use twice for greater effect.
  -P0 Don't ping hosts (needed to scan www.microsoft.com and others)
* -Ddecoy_host1,decoy2[,...] Hide scan using many decoys
  -6 scans via IPv6 rather than IPv4
  -T <Paranoid|Sneaky|Polite|Normal|Aggressive|Insane> General timing policy
  -n/-R Never do DNS resolution/Always resolve [default: sometimes resolve]
  -oN/-oX/-oG <logfile> Output normal/XML/grepable scan logs to <logfile>
  -iL <inputfile> Get targets from file; Use '-' for stdin
* -S <your_IP>/-e <devicename> Specify source address or network interface
  --interactive Go into interactive mode (then press h for help)
```

As you can see there are numerous options that can be used for all types of scans. This paper will look deeper into using the "-D" option. The "-D" option is used for decoy scanning and can help provide some anonymity for an attacker. The idea behind the decoy scan is to flood the logs of the victim systems with attacks appearing to come from a multitude of IP addresses.

The basic idea behind decoy scan is that an attacker will use this option along with a number of IP addresses. Nmap will then scan the target using its real IP and the spoofed IP addresses that were included with the decoy option. This will make it appear to the target system that many systems are port scanning. For example, if an attacker uses the decoy scan with 10 IP addresses, the victim will see 11 addresses scanning it (10 spoofed and 1 attacker). Now whoever is monitoring this system will not be sure which IP address is the real attacking machine. This paper will delve deeper into this and see if anything can be discovered about the scans.

In our testing of the decoy scanning option, we will be using three machines in a 192.168.0.x address space.

Victim - Windows XP – 192.168.0.3
Attacker - FreeBSD 5.2.1 – 192.168.0.6
Decoy - Gentoo Linux – 192.168.0.4

In the example packet dumps below, the timestamps will be colored blue for readability of a new packet and the machine's IP address in question will be colored in red.

Example 1 – All three machines online

In this example we are going to look at using the decoy scan and using a decoy IP address of a machine that is live on the same network.

Command line used:

```
nmap -sS -vv -p 135 -D192.168.0.4 192.168.0.3
```

Note: Decoy scans do not work with bounce or connect scans. Nmap just runs normally but does not spoof any packets if one of those types of scans is chosen. For this reason these examples all use SYN scans.

Below is what the Attacker FreeBSD sniffed using tcpdump during the scan.

```
10:24:01.513777 192.168.0.6 > 192.168.0.3: icmp: echo request (ttl 40, id 25092, len 28)
10:24:01.513840 192.168.0.4 > 192.168.0.3: icmp: echo request (ttl 48, id 61709, len 28)
10:24:01.513900 192.168.0.3 > 192.168.0.6: icmp: echo reply (ttl 128, id 15184, len 28)
10:24:01.513981 192.168.0.6.43145 > 192.168.0.3.80: . [tcp sum ok] 775601886:775601886(0)
ack 3963272926 win 2048 (ttl 57, id 21289, len 40)
10:24:01.514007 192.168.0.4.43145 > 192.168.0.3.80: . [tcp sum ok] 775601886:775601886(0)
ack 3963272926 win 4096 (ttl 51, id 31589, len 40)
10:24:01.514079 192.168.0.3.80 > 192.168.0.6.43145: R [tcp sum ok]
3963272926:3963272926(0) win 0 (ttl 128, id 15186, len 40)
10:24:01.863148 192.168.0.6.43125 > 192.168.0.3.135: S [tcp sum ok]
1673354269:1673354269(0) win 2048 (ttl 37, id 12203, len 40)
10:24:01.863197 192.168.0.4.43125 > 192.168.0.3.135: S [tcp sum ok]
1673354269:1673354269(0) win 4096 (ttl 47, id 64737, len 40)
10:24:01.863289 192.168.0.3.135 > 192.168.0.6.43125: S [tcp sum ok]
3065429684:3065429684(0) ack 1673354270 win 64240 <mss 1460> (DF) (ttl 128, id 15189, len
 44)
```

```
10:24:01.863331 192.168.0.6.43125 > 192.168.0.3.135: R [tcp sum ok]
1673354270:1673354270(0) win 0 (DF) (ttl 64, id 9597, len 40)
```

Here the attacking machine sends a ping from itself and sends a ping using the spoofed IP address of the Decoy machine. The ICMP packets are sent using different TTLs and ID numbers. It then does a type of TCP ping (default action of Nmap) by sending an ACK packet to port 80 using both its real address and spoofing the Decoy address. A RST packet is then sent back to the Attacker machine because that port is closed on the victim machine. We assume that a RST was also sent to the Decoy's machine as well but will look for that later on when we analyze the other systems. In these packets we notice that Nmap is using the same source port, the same sequence number and the same acknowledgement number for the Attacker and the Decoy; however, the window size, TTL and ID number are all different. Next is the scan for port 135 and Nmap sends a SYN packet to port 135 using the Attacker's address and spoofing the Decoy's address. The packets are using the same source port and sequence number, while the window size, TTL and ID field are all different. The Attacker receives a SYN ACK packet back from the victim's machine and finishes by sending a RST packet to the Victim using only the real IP address.

Below is what the Decoy Gentoo machine saw using tcpdump.

```
10:28:15.615276 192.168.0.3 > 192.168.0.4: icmp: echo reply (ttl 128, id 15185, len 28)
10:28:15.615277 192.168.0.3.80 > 192.168.0.4.43145: R [tcp sum ok]
3963272926:3963272926(0) win 0 (ttl 128, id 15187, len 40)
10:28:15.964503 192.168.0.3.135 > 192.168.0.4.43125: S [tcp sum ok]
3065470105:3065470105(0) ack 1673354270 win 64240 <mss 1460> (DF) (ttl 128, id 15190, len
44)
10:28:15.964533 192.168.0.4.43125 > 192.168.0.3.135: R [tcp sum ok]
1673354270:1673354270(0) win 0 (DF) (ttl 64, id 57081, len 40)
```

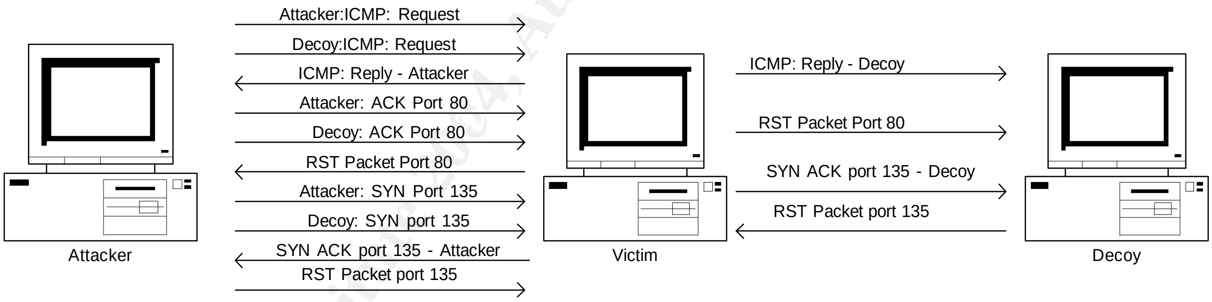
Here the Decoy machine first sees an echo reply to a ping they did not send. Next, the RST packet for the port 80 TCP ping that Nmap performed. Then a SYN ACK packet is received by the Decoy machine after the Attacker has scanned for port 135. Finally the decoy machine responds with a RST packet because it never sent a SYN packet to Victim machine.

Below is the packet dump that the Victim Windows saw captured using ethereal.

```
10:22:09.712321 192.168.0.6 > 192.168.0.3: icmp: echo request (ttl 40, id 25092, len 28)
10:22:09.712348 192.168.0.3 > 192.168.0.6: icmp: echo reply (ttl 128, id 15184, len 28, bad
cksum 0!)
10:22:09.712380 192.168.0.4 > 192.168.0.3: icmp: echo request (ttl 48, id 61709, len 28)
10:22:09.712519 192.168.0.6.43145 > 192.168.0.3.80: . [tcp sum ok] 775601886:775601886(0)
ack 3963272926 win 2048 (ttl 57, id 21289, len 40)
10:22:09.712533 192.168.0.3.80 > 192.168.0.6.43145: R [tcp sum ok]
3963272926:3963272926(0) win 0 (ttl 128, id 15186, len 40)
10:22:09.712550 192.168.0.3 > 192.168.0.4: icmp: echo reply (ttl 128, id 15185, len 28, bad
cksum 0!)
10:22:09.712553 192.168.0.4.43145 > 192.168.0.3.80: . [tcp sum ok] 775601886:775601886(0)
ack 3963272926 win 4096 (ttl 51, id 31589, len 40)
```

```
10:22:09.712560 192.168.0.3.80 > 192.168.0.4.43145: R [tcp sum ok]
3963272926:3963272926(0) win 0 (ttl 128, id 15187, len 40)
10:22:10.061691 192.168.0.6.43125 > 192.168.0.3.135: S [tcp sum ok]
1673354269:1673354269(0) win 2048 (ttl 37, id 12203, len 40)
10:22:10.061737 192.168.0.3.135 > 192.168.0.6.43125: S [tcp sum ok]
3065429684:3065429684(0) ack 1673354270 win 64240 <mss 1460> (DF) (ttl 128, id 15189, len
44, bad cksum 0!)
10:22:10.061756 192.168.0.4.43125 > 192.168.0.3.135: S [tcp sum ok]
1673354269:1673354269(0) win 4096 (ttl 47, id 64737, len 40)
10:22:10.061770 192.168.0.3.135 > 192.168.0.4.43125: S [tcp sum ok]
3065470105:3065470105(0) ack 1673354270 win 64240 <mss 1460> (DF) (ttl 128, id 15190, len
44, bad cksum 0!)
10:22:10.061867 192.168.0.6.43125 > 192.168.0.3.135: R [tcp sum ok]
1673354270:1673354270(0) win 0 (DF) (ttl 64, id 9597, len 40)
10:22:10.061905 192.168.0.4.43125 > 192.168.0.3.135: R [tcp sum ok]
1673354270:1673354270(0) win 0 (DF) (ttl 64, id 57081, len 40)
```

First the victim machine sees a ping from the attacking machine and the decoy machine. These two packets have different TTLs and different ID numbers. Then an Nmap TCP ping ACK packet is sent from both the attacking machine and the decoy machine directed to port 80. We see here again that the source port, sequence number and acknowledgement number are the same, while the window size, TTL and ID number are different.
Here is a quick diagram of the traffic.



Using this same example of all three machines being online, we now scanned for port 150 since we know it is closed. The ICMP and TCP ping are the same as above. The initial SYN packets are the same as well. The difference comes in that the Victim now responds to both the Attacker and the Decoy with a RST ACK because port 150 is closed on the Victim. The other difference in this example is there are no RST packets sent from the Attacker and Decoy IP addresses because there is no need since the RST ACK already tore down the connection.
Another check was to keep all three machines live on the network and scan for multiple ports. The only item noticed here is that the source port stayed the same for both the Attacker and the Decoy machine, even across multiple ports.
Relevant packets are below.

```
12:56:41.308658 192.168.0.6.57925 > 192.168.0.3.135: S [tcp sum ok]
4202031223:4202031223(0) win 2048 (ttl 41, id 4978, len 40)
12:56:41.308707 192.168.0.4.57925 > 192.168.0.3.135: S [tcp sum ok]
4202031223:4202031223(0) win 2048 (ttl 49, id 63568, len 40)
12:56:41.308752 192.168.0.6.57925 > 192.168.0.3.80: S [tcp sum ok]
4202031223:4202031223(0) win 1024 (ttl 52, id 20306, len 40)
12:56:41.308784 192.168.0.3.135 > 192.168.0.6.57925: S [tcp sum ok] 864486713:864486713(0)
ack 4202031224 win 64240 <mss 1460> (DF) (ttl 128, id 19406, len 44)
12:56:41.308834 192.168.0.6.57925 > 192.168.0.3.135: R [tcp sum ok]
4202031224:4202031224(0) win 0 (DF) (ttl 64, id 13020, len 40)
12:56:41.308847 192.168.0.3.80 > 192.168.0.6.57925: R [tcp sum ok] 0:0(0) ack 4202031224
win 0 (ttl 128, id 19408, len 40)
12:56:41.308877 192.168.0.4.57925 > 192.168.0.3.80: S [tcp sum ok]
4202031223:4202031223(0) win 4096 (ttl 39, id 43869, len 40)
12:56:41.308924 192.168.0.6.57925 > 192.168.0.3.139: S [tcp sum ok]
4202031223:4202031223(0) win 1024 (ttl 56, id 25186, len 40)
12:56:41.308967 192.168.0.4.57925 > 192.168.0.3.139: S [tcp sum ok]
4202031223:4202031223(0) win 2048 (ttl 41, id 26117, len 40)
12:56:41.309022 192.168.0.3.139 > 192.168.0.6.57925: S [tcp sum ok] 864603483:864603483(0)
ack 4202031224 win 64240 <mss 1460> (DF) (ttl 128, id 19410, len 44)
12:56:41.309042 192.168.0.6.57925 > 192.168.0.3.139: R [tcp sum ok]
4202031224:4202031224(0) win 0 (DF) (ttl 64, id 13021, len 40)
```

You can see that the source port 192.168.0.6 and 192.168.0.4 are always the same along with the sequence number. The TTLs, window size and ID number are all altered every time. Lets take a quick look at MAC addresses.

```
10:22:09.712321 0:d:61:c2:51:25 0:1:2:45:69:ca 0800 60: 192.168.0.6 > 192.168.0.3: icmp: echo
request
10:22:09.712348 0:1:2:45:69:ca 0:d:61:c2:51:25 0800 42: 192.168.0.3 > 192.168.0.6: icmp: echo
reply
10:22:09.712380 0:d:61:c2:51:25 0:1:2:45:69:ca 0800 60: 192.168.0.4 > 192.168.0.3: icmp: echo
request
10:22:09.712394 0:1:2:45:69:ca ff:ff:ff:ff:ff:ff 0806 42: arp who-has 192.168.0.4 tell 192.168.0.3
10:22:09.712519 0:d:61:c2:51:25 0:1:2:45:69:ca 0800 60: 192.168.0.6.43145 > 192.168.0.3.80: .
ack 3963272926 win 2048
10:22:09.712533 0:1:2:45:69:ca 0:d:61:c2:51:25 0800 54: 192.168.0.3.80 > 192.168.0.6.43145: R
3963272926:3963272926(0) win 0
```

Here we can see that Nmap does not alter the MAC address of the machines as we see in the ICMP echo request packets that both of them have the MAC of the Attacker machine. If the attack happens purely on a local network and there is no other device that packets pass through before reaching the Victim, then the Attacker's real MAC would give them away.

One other test comes to mind and that is using a decoy scan from an outside machine for test purposes. In this test, I use www.google.com (216.239.41.104) as my Decoy. All the traffic seen from the Attacker machine is the same, however there is one difference in what the Victim machine sees.

Below is the traffic dump from the Victim machine.

```
13:17:38.502637 192.168.0.6 > 192.168.0.3: icmp: echo request (ttl 43, id 25954, len 28)
13:17:38.502662 192.168.0.3 > 192.168.0.6: icmp: echo reply (ttl 128, id 20342, len 28, bad
cksum 0!)
13:17:38.502671 216.239.41.104 > 192.168.0.3: icmp: echo request (ttl 52, id 23351, len 28)
```

```
13:17:38.502680 192.168.0.3 > 216.239.41.104: icmp: echo reply (ttl 128, id 20343, len 28, bad
cksum 0!)
13:17:38.502695 192.168.0.6.46676 > 192.168.0.3.80: . [tcp sum ok]
1422159262:1422159262(0) ack 3825495454 win 2048 (ttl 45, id 14016, len 40)
13:17:38.502709 192.168.0.3.80 > 192.168.0.6.46676: R [tcp sum ok]
3825495454:3825495454(0) win 0 (ttl 128, id 20344, len 40)
13:17:38.502724 216.239.41.104.46676 > 192.168.0.3.80: . [tcp sum ok]
1422159262:1422159262(0) ack 3825495454 win 1024 (ttl 40, id 33826, len 40)
13:17:38.502730 192.168.0.3.80 > 216.239.41.104.46676: R [tcp sum ok]
3825495454:3825495454(0) win 0 (ttl 128, id 20345, len 40)
13:17:38.852172 192.168.0.6.46652 > 192.168.0.3.135: S [tcp sum ok]
3635330667:3635330667(0) win 4096 (ttl 59, id 15551, len 40)
13:17:38.852218 192.168.0.3.135 > 192.168.0.6.46652: S [tcp sum ok]
1178484739:1178484739(0) ack 3635330668 win 64240 <mss 1460> (DF) (ttl 128, id 20347, len
44, bad cksum 0!)
13:17:38.852236 216.239.41.104.46652 > 192.168.0.3.135: S [tcp sum ok]
3635330667:3635330667(0) win 4096 (ttl 39, id 37052, len 40)
13:17:38.852250 192.168.0.3.135 > 216.239.41.104.46652: S [tcp sum ok]
1178531483:1178531483(0) ack 3635330668 win 64240 <mss 1460> (DF) (ttl 128, id 20348, len
44, bad cksum 0!)
13:17:38.852349 192.168.0.6.46652 > 192.168.0.3.135: R [tcp sum ok]
3635330668:3635330668(0) win 0 (DF) (ttl 64, id 13834, len 40)
13:17:41.912607 192.168.0.3.135 > 216.239.41.104.46652: S [tcp sum ok]
1178531483:1178531483(0) ack 3635330668 win 64240 <mss 1460> (DF) (ttl 128, id 20370, len
44, bad cksum 0!)
```

The major difference here is that Google is not responding with a RST when it receives the SYN ACK from the Victim machine. This is probably due to the network configuration on Google's side where uninitiated SYN ACK packets are just dropped. This appears to be one way that the Victim would be able to identify the real attacker. If Google had initiated the scan the SYN would have been saved in its network perimeter's state table and the SYN ACK would have been allowed to reach the machine and the RST would have followed. Since the Victim machine only received a RST from 192.168.0.4, it should be the real attacker.

Learned from this event:
It does appear that we will be able to tell when we have been scanned using the decoy option of Nmap. The signs to look for would be multiple IP addresses doing the exact same thing with the same source ports, same sequence numbers and same acknowledgement number. The hard part is telling which one of these performed the real scan. Looking at the information presented so far, it does not likely to be able to see who it was the guilty party. Nmap is smart enough to spoof both the ICMP ping and the TCP ping using both addresses and even sends the RST packet from the Attacker machine because it knows that the Decoy machine will send a RST. Another note is if the attacker turns pings off, they would be even quieter because the ICMP and TCP ping traffic would be gone above. The ping was left on in these examples to see how Nmap would handle spoofing them.
One issue that was brought up is that the attacker should pick their decoys carefully and be fully aware of their responses. Nmap makes no contact with

decoy machines and expects the decoy to send back a RST packet. If the decoy does not send the final RST, it can be a give away to the real attacker. Also, if the packets do not pass through any other devices, then the true MAC address of the attacker would be seen. An insider on a network with a network device between them and the Victim would be able to scan with anonymity. The key to any of the decoy scanning working is the choosing of the decoys. If the attacker used a number of decoys, it would take an administrator a good amount of time to track down each owner of the Decoy machines.

Example 2 – Victim and Attacker online with Decoy offline

In this example we are going to analyze what takes place if the Decoy machine is not online.

Command line used:

```
nmap -sS -vv -p 135 –P0 -D192.168.0.4 192.168.0.3
```

Below is the traffic dump from the Attacker machine.

```
13:05:56.791545 192.168.0.6 > 192.168.0.3: icmp: echo request (ttl 46, id 13867, len 28)
13:05:56.791605 192.168.0.4 > 192.168.0.3: icmp: echo request (ttl 38, id 30286, len 28)
13:05:56.791666 192.168.0.3 > 192.168.0.6: icmp: echo reply (ttl 128, id 19754, len 28)
13:05:56.791747 192.168.0.6.59898 > 192.168.0.3.80: . [tcp sum ok]
3971623390:3971623390(0) ack 3174705630 win 4096 (ttl 47, id 56817, len 40)
13:05:56.791790 192.168.0.4.59898 > 192.168.0.3.80: . [tcp sum ok]
3971623390:3971623390(0) ack 3174705630 win 3072 (ttl 42, id 23427, len 40)
13:05:56.791847 192.168.0.3.80 > 192.168.0.6.59898: R [tcp sum ok]
3174705630:3174705630(0) win 0 (ttl 128, id 19756, len 40)
13:05:57.139300 192.168.0.6.59874 > 192.168.0.3.135: S [tcp sum ok]
2543895146:2543895146(0) win 4096 (ttl 59, id 5518, len 40)
13:05:57.139348 192.168.0.4.59874 > 192.168.0.3.135: S [tcp sum ok]
2543895146:2543895146(0) win 3072 (ttl 46, id 62662, len 40)
13:05:57.139431 192.168.0.3.135 > 192.168.0.6.59874: S [tcp sum ok] 991935949:991935949(0)
ack 2543895147 win 64240 <mss 1460> (DF) (ttl 128, id 19766, len 44)
13:05:57.139472 192.168.0.6.59874 > 192.168.0.3.135: R [tcp sum ok]
2543895147:2543895147(0) win 0 (DF) (ttl 64, id 13325, len 40)
```

The attacker machine sees the exact same traffic that was seen in example 1. There are no differences discovered.

Below is the traffic dump from the Victim machine.

```
13:04:04.111732 192.168.0.6 > 192.168.0.3: icmp: echo request (ttl 46, id 13867, len 28)
13:04:04.111756 192.168.0.3 > 192.168.0.6: icmp: echo reply (ttl 128, id 19754, len 28, bad
cksum 0!)
13:04:04.111788 192.168.0.4 > 192.168.0.3: icmp: echo request (ttl 38, id 30286, len 28)
13:04:04.111932 192.168.0.6.59898 > 192.168.0.3.80: . [tcp sum ok]
3971623390:3971623390(0) ack 3174705630 win 4096 (ttl 47, id 56817, len 40)
13:04:04.111945 192.168.0.3.80 > 192.168.0.6.59898: R [tcp sum ok]
3174705630:3174705630(0) win 0 (ttl 128, id 19756, len 40)
13:04:04.111972 192.168.0.4.59898 > 192.168.0.3.80: . [tcp sum ok]
3971623390:3971623390(0) ack 3174705630 win 3072 (ttl 42, id 23427, len 40)
13:04:04.459487 192.168.0.6.59874 > 192.168.0.3.135: S [tcp sum ok]
2543895146:2543895146(0) win 4096 (ttl 59, id 5518, len 40)
13:04:04.459530 192.168.0.3.135 > 192.168.0.6.59874: S [tcp sum ok] 991935949:991935949(0)
ack 2543895147 win 64240 <mss 1460> (DF) (ttl 128, id 19766, len 44, bad cksum 0!)
```

```
13:04:04.459546 192.168.0.4.59874 > 192.168.0.3.135: S [tcp sum ok]
2543895146:2543895146(0) win 3072 (ttl 46, id 62662, len 40)
13:04:04.459657 192.168.0.6.59874 > 192.168.0.3.135: R [tcp sum ok]
2543895147:2543895147(0) win 0 (DF) (ttl 64, id 13325, len 40)
```

In this example it is easy to see who the real Attacker was because the Victim issues an arp request (not shown in packet dump) for Decoy and receives no response so it no longer sends traffic back out to the Decoy machine. The ICMP and TCP ping and the SYN packet directed at port 135 from the Decoy machine all reach the Victim machine, but the Victim machine does not know how to reach it because of the failed arp request so there only replies made to the Attacker. To avoid the arp issue, the next example spoofs an outside IP address of a non responding host. An outside address was selected from my Apache logs that is not currently responding to anything. Once again in this example no RST packet is sent back because the machine is not alive. This again shows the importance of the decoy IP addresses used.

Lessons Learned:
The “-D” option for Nmap, when used properly, can be very effective for performing port scans with some anonymity. It is not complete anonymity because the Attacker still makes contact, but it is an attempt to cause more work then an administrator is willing to put forward to track down a port scan. These examples above are very small examples with only one decoy being used, however it is imaginable how large these dumps could grow if numerous decoys were used. If an attacker used 40 well-chosen decoys, it would cost much time to track down all the decoys and it would be extremely difficult to increase your monitoring on all of those IP addresses. The only easy way for the real Attacker to be discovered using only captured traffic would be to look for RST packets that are sent. Everything else is spoofed or random. Ping traffic is all spoofed and TTLs are random so attempting to trace it back could be difficult. Also one trend that showed up in the examples here was that the legitimate attacker always seemed to be the first IP address to send a ping to the Victim. However, after running numerous tests using multiple decoys, this was not found to be the case. Nmap randomizes which source IPs are using with pings are turned on or off. The last thing that was notice when going back over all the examples is that the RST packet sent from the Attacker does not use an altered TTL. This might be because the decoys are going to send RST packets using their default TTL and this way the Attacker seems the same. Going from all fake to all real appears to keep everything anonymous still.

References

Fyodor, “Nmap Free Security Scanner”, URL http://www.insecure.org/

Moore, HD, “Nmap Development: Re: Finding real host in Nmap –D Scans”, URL http://seclists.org/lists/nmap-dev/2003/Jan-Mar/0043.html (March 2003)

Morris, Gary, “NMAP Decoy Scan?”, URL http://cert.uni-stuttgart.de/archive/intrusions/2002/09/msg00173.html (September 2002)

Vision, Max, “NMAP: Decoy Analysis”, URL http://www.inet-sec.org/docs/scanning/nmapdecoy.html (April 1999)

Whistler, “Camouflaging Nmap Scans”, URL http://www.hackinthebox.org/article.php?sid=10640

Part 2

First detect – Backdoor Q Access

1. Source of Trace:

This trace came from: http://www.incidents.org/logs/Raw/2002.9.31
The trace came from incidents.org so a positive layout of the network is not possible. However, Pete Storm did an excellent analysis in his practical over what he believes the layout of the network to look like. After going through his analysis, I find no reason to disagree with his ideas. I used his methodology to outline the network (http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf [11]). Below are my findings using his methodology.
Searching the file for source MAC addresses shows only two addresses existing:

```
tcpdump -ner 2002.9.31 | awk '{print $2}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

Searching for destination MAC addresses shows the same two.

```
tcpdump -ner 2002.9.31 | awk '{print $3}' | sort -u
0:0:c:4:b2:33
0:3:e3:d9:26:c0
```

I checked these MACs at http://standards.ieee.org/regauth/oui/index.shtml and they showed that both of these MACs do belong to Cisco.
The next step is to identify the IP addresses associated with each NIC.
The source IP addresses associated with 0:0:c:4:b2:33 are found using:

```
tcpdump -ner 2002.9.31 ether src 0:0:c:4:b2:33 | awk '{print $6}' | awk -F \. '{print $1 "." $2 "." $3
"." $4}' | sort -u
207.166.87.157
207.166.87.40
```

Next we see where are packets associated with MAC 0:0:c:4:b2:33 are destined.

```
tcpdump -ner 2002.9.31 ether src 0:0:c:4:b2:33 | awk '{print $8}' | awk -F \. '{print $1 "." $2 "." $3
"." $4}' | sort –u
12.141.80.145
12.145.6.137
12.163.48.217
12.164.249.178
…
81.96.182.183
81.96.84.47
81.97.220.59
81.98.80.68
```

Next we see what IP addresses associate with MAC 0:3:e3:d9:26:c0 as a source.

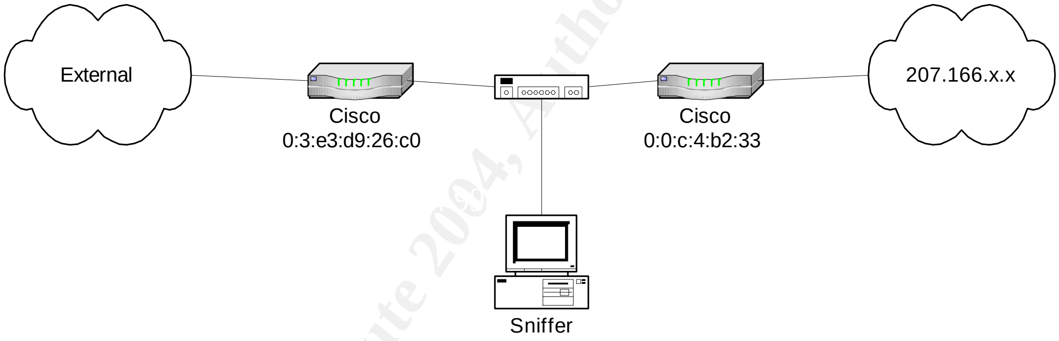
```
tcpdump -ner 2002.9.31 ether src 0:3:e3:d9:26:c0 | awk '{print $6}' | awk -F \. '{print $1 "." $2 "."
$3 "." $4}' | sort –u
12.111.47.194
128.167.120.13
```

```
128.167.69.13
129.174.184.87
…
80.67.68.16
80.67.68.8
81.19.69.28
81.89.69.194
```

Now we find out where packets associated with MAC 0:3:e3:d9:26:c0 are destined.

```
tcpdump -ner 2002.9.31 ether src 0:3:e3:d9:26:c0 | awk '{print $8}' | awk -F \. '{print $1 "." $2 "."
$3 "." $4}' | sort –u
207.166.0.99
207.166.1.137
207.166.1.186
207.166.10.121
…
207.166.96.128
207.166.97.78
207.166.98.212
207.166.98.238
```

From that we can get an idea of network setup.



2. Detect generated by:

The raw file was run through Snort v 2.1.1 using default rules and a default configuration file except all rule files were turned on.

The rule the triggered the event was

```
alert tcp 255.255.255.0/24 any -> $HOME_NET any (msg:"BACKDOOR Q access"; flags:A+;
dsize: >1;  stateless; reference:arachnids,203; sid:184;  classtype:misc-
activity; rev:4;)
```

Here are example packets:

```
19:53:10.796507 255.255.255.255.31337 > 207.166.124.130.515: R [bad tcp cksum b5b5!] 0:3(3)
ack 0 win 0 [RST cko] (ttl 15, id 0, len 43, bad cksum a9ef!)
0x0000   4500 002b 0000 0000 0f06 a9ef ffff ffff        E..+............
0x0010   cfa6 7c82 7a69 0203 0000 0000 0000 0000  ..|.zi..........
0x0020   5014 0000 5f17 0000 636b 6f00 0000          P..._...cko...
```

```
--
20:04:49.846507 255.255.255.255.31337 > 207.166.218.167.515: R [bad tcp cksum b5b5!] 0:3(3)
ack 0 win 0 [RST cko] (ttl 15, id 0, len 43, bad cksum 4bca!)
0x0000   4500 002b 0000 0000 0f06 4bca ffff ffff        E..+......K.....
0x0010   cfa6 daa7 7a69 0203 0000 0000 0000 0000   ....zi..........
0x0020   5014 0000 00f2 0000 636b 6f00 0000           P.......cko...
--
20:29:43.876507 255.255.255.255.31337 > 207.166.133.190.515: R [bad tcp cksum b5b5!] 0:3(3)
ack 0 win 0 [RST cko] (ttl 15, id 0, len 43, bad cksum a0b3!)
0x0000   4500 002b 0000 0000 0f06 a0b3 ffff ffff        E..+............
0x0010   cfa6 85be 7a69 0203 0000 0000 0000 0000   ....zi..........
0x0020   5014 0000 55db 0000 636b 6f00 0000          P...U...cko...
--
21:47:19.006507 255.255.255.255.31337 > 207.166.185.185.515: R [bad tcp cksum b5b5!] 0:3(3)
ack 0 win 0 [RST cko] (ttl 15, id 0, len 43, bad cksum 6cb8!)
0x0000   4500 002b 0000 0000 0f06 6cb8 ffff ffff        E..+......l.....
0x0010   cfa6 b9b9 7a69 0203 0000 0000 0000 0000   ....zi..........
0x0020   5014 0000 21e0 0000 636b 6f00 0000           P...!...cko...
```

Here we can see why the rule was fired. It is TCP traffic with a source address of 255.255.255.255 and the rule is looking for this source address. The rule is looking for any source port, any destination IP address and any destination port so those in the actual packet do not matter when it comes to triggering the rule. The rule also checks for the Ack flag being set, which in our example packet it is set.

Here is the breakdown of one of the example packets.

		Notes:
IP version	4	
IP header Length	5	20 bytes
Total Datagram Length	2b	43 bytes = 20 bytes IP header + 20 bytes TCP header + 3 bytes payload
Id	0	Is 0 for all 46 alerts
TTL	0f	0f = 15 - All the same for 46 alerts
Protocol	6	TCP
Source IP	255.255.255.255	ffffffff; Broadcast address
Destination IP	207.166.185.185	cfa6 daa7
Source Port	31337	7A69; Hacker speak for 'eleet'
Destination Port	515	0203
TCP Flags	Rst, Ack	Ack number is 0 which is not possible unless crafted packet

Here is what Snort shows us:

```
[**] [1:184:4] BACKDOOR Q access [**]
[Classification: Misc activity] [Priority: 3]
10/30-23:59:13.196507 255.255.255.255:31337 -> 207.166.177.167:515
TCP TTL:15 TOS:0x0 ID:0 IpLen:20 DgmLen:43
***A*R** Seq: 0x0 Ack: 0x0 Win: 0x0 TcpLen: 20
[Xref => http://www.whitehats.com/info/IDS203]
```

Log format
The Snort output shows you the name of the rule that has been violated, the classification of the alert and the priority. Next is the date and time stamp followed by the source IP address and source port. Next is the destination IP address and destination port. After that more detailed information about the packet including flags, TTL, etc. Finally the last line is a URL to reference information about the alert.

3. Probability the source address was spoofed:

The IP address is definitely spoofed because 255.255.255.255 is a broadcast address. An attacker spoofing this IP would not see any response from their traffic unless they were on the same subnet because any traffic sent back to 255.255.255.255 would not leave the local subnet. Using Reset and Ack flags could be an attempt to bypass firewalls that may allow this type of traffic through. Destination port 515 could also be open to firewalls because it is the LDP printer port that many Unix boxes have open. A poorly configured firewall may allow traffic to this port. Another sign that this is a crafted packet is the source port is 31337 – hacker speak for "eleet" like elite hacker. Notice the ID fields are always the same and all the TTLs are the same, indicating it is possible that these packets could be coming from the same attacker.

4. Description of attack:

This appears to be an attacker attempting to send commands to a backdoor Trojan that has been installed on the destination IP. This Trojan could be one that they setup or more likely, they are looking to find a system infected with the Trojan. Snort recognized this as Q-Backdoor traffic. Q was written by a person named Mixter (http://mixter.void.ru/) that created the remote control tool. Les Gordon gives a complete analysis of the Q Trojan at http://www.sans.org/resources/idfaq/qtrojan.php. Q works on a client/server architecture and attempts to be a secure remote administration tool by incorporating strong encryption. The interesting part about the tool is that it can be controlled using raw ICMP, TCP or UDP and no response to the sender is necessary. Since the server uses raw sockets, destination ports would not matter because it sets the NIC to listen in promiscuous mode which means that it can sniff all traffic on the wire that the infected machine can see. For example, the remote user could use the client program to send a TCP packet using any source address and any destination port. In the packet itself would be the command (possibly encrypted) that the remote user wanted to run. Using this method the attacker could send one packet to the server and request that the server ping a certain IP address that was logging pings to create a list of compromised systems. This is just an example of what could be done and a malicious attacker could run any number of commands. This tool allows for anonymity in controlling the boxes since no established connection is necessary for communicating with the server and all traffic can be spoofed. The port of 515 was possibly picked hoping that it would be allowed into the network because it is a printer port for LPD running on UNIX and Linux and the firewall may have it opened. It is an interesting choice because there are many other ports that are more likely to be open - port 80, 25, or 53 for example. Also if this is the Q Trojan, it is an older version because new versions use random source addresses and only in the older versions could you specify a source IP address. This traffic appears to have been set with a source since all alerts use the same source IP address.

This alert alone is not proof of the Q Trojan. This alert is generic enough that other trojans or crafted packets could set it off. The broadcast address is an easy address to spoof and could be used in other malicious traffic.

5. Attack mechanism:

If this is the Q Trojan, then this traffic would be considered a stimulus. The Q Trojan sends a command to a destination computer. If that machine is infected with the Q Trojan it will run the command that was sent to it. If a machine is infected, it should be considered a complete compromise and the machine should be rebuilt since it would be unknown what an attacker may have done to the system.

6. Correlations:

All of the alerts correlate to each other. All the TTLS are the same at 15 and this is a sign that all the packets could have originated from the same source. Other signs are that the source port does not change, the destination port does not change and the ID fields are all the same. The ID field is also set to zero in all the packets which is another sign of crafted packets.

It is difficult to tell if the real source launched anything else our way, since the source is spoofed. None of the destinations had any other alerts triggered so it appears that nothing else was sent there way during this time frame. No other traffic in this capture appears to be related to these 46 alerts.

This event has been analyzed in many other GCIA papers. Mike Wyman brought up an interesting point in his analysis (http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00509.html) in that it would be very rare to see a packet with a Reset flag sent to port 515. This would mean that the LPD server initiated the connection to port 31337. For this to be the case, the LDP service would have to be badly misconfigured.

7. Evidence of active targeting:

The traffic is directed at 45 total hosts with one host being checked twice. The randomness of the IP addresses gives the argument that this is not a directed attack. The source attacks many IP addresses over a 22 hour period and there are no timing relations in the alerts. The times appear completely random making it appear as though it is a scanner that is checking random IP addresses for a backdoor.

8. Severity:

This is difficult for this network since I do not know the systems that are here because these are logs taken from SANS.

Criticality: 5 (unknown)
It is hitting a range of systems and some of those are probably critical

Lethality: 5 (unknown)
This is for the possibility that a backdoor is installed on one of these systems. If this attack reaches one of those machines, the possibilities are endless for the hacker.

System Countermeasures: 2 (unknown)
The systems are unknown to me.

Network Countermeasures: 2 (unknown)
The network prevention is unknown to me. It is possible that the machine used to gather these logs sets in front of a firewall and all this traffic could have been blocked by that.

Total Severity = (5+5) – (2+2) = 6

9. Defensive recommendation:

Only allow necessary ports to be opened at perimeter firewall or edge router. Port 515 is normally reserved for the LPD service and should not need to be used by outside users. If this port must be open for a reason, then an access control list (ACL) should be used to limit the IP addresses that can reach that port. The ACL should be locked down to source IP address, destination IP address and destination port. The perimeter defense should also block specific IP addresses from entering the network and 255.255.255.255 should be one of those. No traffic with that source IP address should make it inside the network. Systems should also be running up-to-date anti-virus software and a file integrity checker that monitors for altered files.

10. Multiple choice test question:

When communicating with the Q Trojan, the attacker must be able to communicate with what destination port on the infected machine?

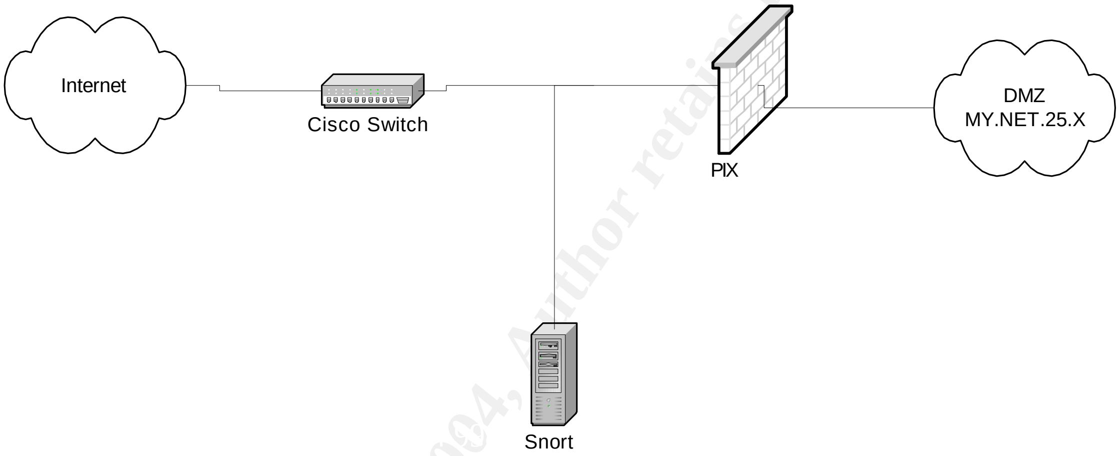
A) 515
B) 80
C) 31337

D) Any
Answer: D

Second detect –Backdoor Typot Trojan Traffic

1. Source of Trace:

This is a trace from a tcpdump file that was created by sniffing a financial institution's network for a business day and then run through Snort for alerting. Data that does not affect the analysis has been altered.
Here is a general layout of the network.



2. Detect was generated by:

Snort v 2.1.1 running on FreeBSD 5.2.1 running default rules and a default configuration will all rules turned on.
Below is the rule that triggered the alert for this detect:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"BACKDOOR typot trojan traffic";
stateless; flags:S,12; window:55808; classtype:trojan-activity; sid:2182; rev:3;)
```

Here are the tcpdump packets:

```
11:26:49.158669 24.15.57.42.22273 > MY.NET.25.197.45251: S [bad tcp cksum b5b5!]
3099421657:3099421657(0) win 55808 <mss 1460,nop,wscale 2,nop,nop,sackOK> (ttl 115,id
49494, len 52)
0x0000   4500 0034 c156 0000 7306 ffff 180f 392a        E..4.V..s.=...9*
0x0010   c0a8 19c5 5701 b0c3 b8bd 6bd9 0000 0000        .l(.W.....k.....
0x0020   8002 da00 1f4f 0000 0204 05b4 0103 0302        .....O..........
0x0030   0101 0402                                      ....

11:48:31.353099 37.145.195.236.22273 > MY.NET.25.197.45251: S [bad tcp cksum b5b5!]
3099421657:3099421657(0) win 55808 <mss 1460,nop,wscale 2,nop,nop,sackOK> (ttl 114, id
28152, len 52)
```

```
0x0000   4500 0034 6df8 0000 7206 f91c 2591 c3ec        E..4m...r...%...
0x0010   c0a8 19c5 5701 b0c3 b8bd 6bd9 0000 0000        .I(.W.....k.....
0x0020   8002 da00 870a 0000 0204 05b4 0103 0302        ................
0x0030   0101 0402                                       ....
```

Quick breakdown of the packet:

		Notes:
IP version	4	
IP header Length	5	20 bytes
Total Datagram Length	34	52 bytes = 20 IP header + 32 TCP Header
Id	c156	49494
TTL	73	115
Protocol	6	TCP
Source Port	5701	22273
Destination Port	b0c3	45251
TCP Flags	Syn	

Here is what Snort shows us:

```
 [**] [1:2182:3] BACKDOOR typot trojan traffic [**]
   [Classification: A Network Trojan was detected] [Priority: 1]
   04/05-11:26:49.158669 24.15.57.42:22273 -> MY.NET.25.197:45251
   TCP TTL:115 TOS:0x0 ID:49494 IpLen:20 DgmLen:52
   ******S* Seq: 0xB8BD6BD9 Ack: 0x0 Win: 0xDA00 TcpLen: 32
   TCP Options (6) => MSS: 1460 NOP WS: 2 NOP NOP SackOK

[**] [1:2182:3] BACKDOOR typot trojan traffic [**]
   [Classification: A Network Trojan was detected] [Priority: 1]
   04/05-11:48:31.353099 37.145.195.236:22273 -> MY.NET.25.197:45251
   TCP TTL:114 TOS:0x0 ID:28152 IpLen:20 DgmLen:52
   ******S* Seq: 0xB8BD6BD9 Ack: 0x0 Win: 0xDA00 TcpLen: 32
   TCP Options (6) => MSS: 1460 NOP WS: 2 NOP NOP SackOK
```

Log format

The Snort output shows you the name of the rule that has been violated, the classification of the alert and the priority. Next is the date and time stamp followed by the source IP address and source port. Next is the destination IP address and destination port. After that more detailed information about the packet including flags, TTL, etc.

3. Probability the source address was spoofed:

The probability is high that this would be a spoofed source address if this is truly the typot Trojan. Being TCP traffic, the initial idea is that it would not be spoofed because TCP communications require the three-way handshake to communicate. However, the ‘typot’ Trojan will send out SYN packets using spoofed IP addresses. It will also listen for incoming SYN packets with certain properties and save these packets to a file. It does this by placing the NIC of the

infected machine in promiscuous mode and that way it sees all traffic that the infected machine can see. For this reason, a three-way handshake does not need to take place and that allows for the spoofing of the source IP address and then also destination ports do not matter because it can be any port.
Another sign that the sources are spoofed IP addresses, is that it appears that the above packets came from the same host even though both source IP addresses are different. The source port, destination IP, destination port, initial sequence number and ID number are all the same. The TTL is off by one but that is close enough to make an educated guess that it possibly came from the same source.

4. Description of attack:

This website includes a good description of this trojan: http://hq.mcafeeasap.com/dispTrojan.asp?virus_k=100406. The 'typot' Trojan is an ELF binary that is compiled for the Linux operating system. The Trojan uses the libnet library to craft and send packets and uses the libpcap library to capture network traffic. When the Trojan is run, it checks the UID to see if it is 'root' and if so it will enter a loop where it sends out SYN packets with a window size of 55808 bytes every second. It spoofs the source IP address and sends the SYN packets to random IP addresses. The source hardware MAC is 'D:E:A:D:0:0 and the destination hardware MAC is 0:10:67:0:b1:86. The port numbers on both sides are also randomly picked. The Trojan will set the network card into promiscuous mode and sniff for incoming SYN packets that have a window size of 55808 bytes and will log the packets in a file named 'r'. After 24 hours the Trojan will attempt to send the captured packets to the IP address 12.108.65.76 and after that it will begin its loop again. If it cannot connect to the IP address above, it will delete the file /tmp/…/a where the Trojan is located.

5. Attack mechanism:

In this detect, this traffic would be considered a stimulus. It is an incoming packet with a window size of 55808 bytes. This alert is not a concern to my network because the destination IP address is a Windows machine and would therefore not be affected by the typot Trojan and my firewall does not allow traffic through on this port. No specific service is being targeted here because it is looking for an infected machine that would be sniffing all traffic directed at the box.
There is one interesting element of this alert and the two alerts share a lot in common. The alerts are separated by about 22 minutes and the source IP address is different; however, the source port, destination IP address, destination port and sequence number are all the same. The TTL is also only off by one. This makes the traffic appear as if it might have come from the same host. I would see this as an infected machine that is sending out the SYN scans and my machine was somehow targeted twice. No other traffic from any of these IP addresses was recorded.

6. Correlations:

This traffic was first noticed back in May 2003 when a security analyst working for the Department of Defense discovered it (http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00153.html). This was seen to be a Trojan and is believed that the traffic in the SYN packets is encrypted instructions for communication. The Trojan's purpose and distribution are still unknown. It is thought to have been an attempt at a very stealthy port scanner and was merely a proof of concept application. It was very inefficient and contained no self-replicating or malicious behavior. Anti-virus vendors went on to name this the 'typot' Trojan. I was unable to find any other analysis in a GCIA practical.

7. Evidence of active targeting:

This traffic does not appear to be actively targeting machines and appears to be completely random traffic. The source IP addresses only sent this one packet into my network and nothing else came from them. Analyzing the packets leads me to believe that it is typot traffic and then it would be random IP addresses that it is targeting.

8. Severity:

Criticality: 3
This particular IP address is a web server that is the front-end to my company. This server is not extremely critical because no sensitive data is stored on it; however, it is what customers see if they visit my site online.

Lethality: 1
This Trojan is non-destructive and would only add to traffic on my network causing a slight increase in bandwidth.

System Countermeasures: 5
The machine is running a Windows operating system that is not affected by the 'typot' Trojan. It also has anti-virus software running that is kept updated on a daily basis.

Network Countermeasures: 5
My perimeter firewall would not allow this traffic to even reach the destination IP address. That server is a Windows web server that only allows traffic over destination port 80. Port 45251 would be blocked after at the firewall.

Total Severity = (2+1) – (5+5) = -7 which makes it a very low severity

9. Defensive recommendation:

In this particular incident, more defense could be done by using a host-based integrity checker like Tripwire. Having this type of host-based IDS, file changes would be noted and could be checked into further. The rest of the defenses that are already in place appear to be adequate in this situation.

10. Multiple choice test question:

The ‘typot’ Trojan looks for a packet with a windows size of:

A) 58808
B) 65535
C) 55808
D) 0
Answer: C

Submittal to Incidents.org mailing list:

This detect was submitted to the incidents.org mailing list twice - once on April 13, 2004 and another time on April 24, 2004. Only two responses were received so those two are post below with my answers. A third response was not received so I was unable to post it here.

Response #1

```
>From: "Joe Bowling" <joebowling@comcast.net>
>Reply-To: "Intrusions List (GCIA Practicals)" <intrusions@lists.sans.org>
>To: "Intrusions List (GCIA Practicals)" <intrusions@lists.sans.org>,
><intrusions@incidents.org>
>Subject: Re: [Intrusions] LOGS: GIAC GCIA Version 3.4 Practical
>(PaulSchmelzel)
>Date: Mon, 26 Apr 2004 00:19:04 -0400
>List-Id: "Intrusions List (GCIA Practicals)" <intrusions.lists.sans.org>
>List-Archive: <http://www.dshield.org/pipermail/intrusions>
>List-Post: <mailto:intrusions@lists.sans.org>
>List-Help: <mailto:intrusions-request@lists.sans.org?subject=help>
>Errors-To: intrusions-bounces@lists.sans.org
>Return-Path: intrusions-bounces@lists.sans.org
>X-OriginalArrivalTime: 26 Apr 2004 11:44:06.0125 (UTC)
>FILETIME=[C78419D0:01C42B83]
>
>
>First i will say excellent job on this detect.
>
>Only question i have is are the TCP options also crafted with the typot
>traffic activity?
>If not then possibly you could attempt to fingerprint the OS of the Source
>IP based on the TCP options (verify source as Linux).
```

>Toby Miller has a great paper on OS fingerprinting based on the syn packet
>only.
>
>Toby the link i found for you paper via goggle is currently broken....can
>you post a good link for it.
>
>Thanks
>
>Best
>
>Joe

Answer to Response #1

I am not sure if the TCP options are crafted by the typot trojan or not. There was no technical breakdown thorough enough that went deep into the packet. I searched on the Internet for similar packet dumps with a window size of 55808 and I found some with similarities to what I am seeing here. Two posts (http://lists.virus.org/snort-users-0403/msg00452.html and http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00164.html) show similar packet dumps with a window size of 55808 with similar TCP options. In the first link, the MSS is 1452 where I saw 1460 but the other options are the same. The second link shows a packet with the exact same options and they are looking at it as some type of scan.

I also attempted a passive fingerprint of the operating system using Toby Miller's paper (http://www.sans.org/rr/special/passiveos2.php). The first process was done by hand and no match was discovered. The TTL makes it appear that it is coming from a Windows machine but none of the other options match with Windows. The rest of the options are closer matched to Linux and BSD defaults. Next, I ran the packets through p0f (http://lcamtuf.coredump.cx/p0f.shtml) but it came back with "Unknown" for those IP addresses.

Response #2

>From: "Johnny Wong" <deepcracksg@yahoo.com.sg>
>Reply-To: "Intrusions List (GCIA Practicals)" <intrusions@lists.sans.org>
>To: "Intrusions List (GCIA Practicals)" <intrusions@lists.sans.org>,
><intrusions@incidents.org>
>Subject: Re: [Intrusions] LOGS: GIAC GCIA Version 3.4 Practical
>(PaulSchmelzel)
>Date: Mon, 26 Apr 2004 09:58:07 +0800
>List-Id: "Intrusions List (GCIA Practicals)" <intrusions.lists.sans.org>
>Errors-To: intrusions-bounces@lists.sans.org
>Return-Path: intrusions-bounces@lists.sans.org
>X-OriginalArrivalTime: 26 Apr 2004 11:41:27.0932 (UTC)
>FILETIME=[6939C3C0:01C42B83]
>

>Hi there,
>
>The Seq numbers and TCP options are also the same, further justifying that
>these two packets could have originated from the same source machine. Would
>u be able to fingerprint the OS of this machine? Just to verify that it is
>indeed a compromised Linux machine.
>
>You mentioned that it was highly likely the IP addresses were spoofed.
>Could
>you co-relate the TTL value with the resolution (lookup) of the source IP?
>One of them could be legitimate.
>
>You missed out the multiple choice question.
>
>Rgds,
>
>Johnny Wong

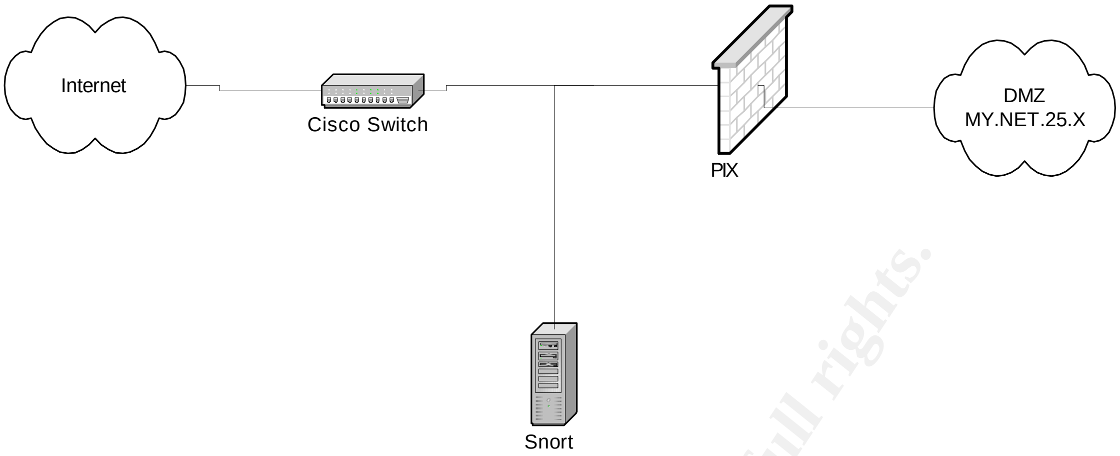
Answer to Response #2

The first time through my analysis I failed to mention that the TCP options were the same and this does add even more argument that these came from the same host. I attempted to do a passive operating system fingerprint but nothing matched up. On your recommendation, I did lookup both source IP addresses but the TTLs do not match either of them either. This lends more support to both of these source IP addresses being spoofed.

Third detect – Scan Proxy Port 8080

1. Source of Trace:

This trace comes from the same network used in the second detect above. It was captured using tcpdump.



2. Detect generated by:

Snort v 2.1.1 running on FreeBSD 5.2.1 using default rules and a default configuration file with all rules turned on.

Here is the Snort rule that triggered this alert:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET 8080 (msg:"SCAN Proxy Port 8080 attempt";
stateless; flags:S,12; classtype:attempted-recon; sid:620; rev:6;)
```

Here is an example tcpdump packet:

```
10:57:37.555389 67.234.73.55.0 > MY.NET.25.78.8080: S [tcp sum ok] 11429793:11429793(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937        E..(.L@.q...C.I7
0x0010   c0a8 194e 0000 1f90 00ae 67a1 0000 0000          .l(N......g.....
0x0020   5002 0200 a127 0000 0000 0000 0000               P....'........
```

Here is what Snort shows us:

```
04/05-10:57:37.555389 67.234.73.55:0 -> MY.NET.25.78:8080
TCP TTL:113 TOS:0x0 ID:844 IpLen:20 DgmLen:40 DF
******S* Seq: 0xAE67A1  Ack: 0x0  Win: 0x200  TcpLen: 20
```

Here is a breakdown of the example packet:

		Notes:
IP version	4	
IP header Length	5	20 bytes
Total Datagram Length	28	40 bytes = 20 IP header + 20 TCP Header
Id	034c	844
TTL	71	113
Protocol	06	TCP
Source Port	0	0
Destination Port	1f90	8080
TCP Flags	Syn	

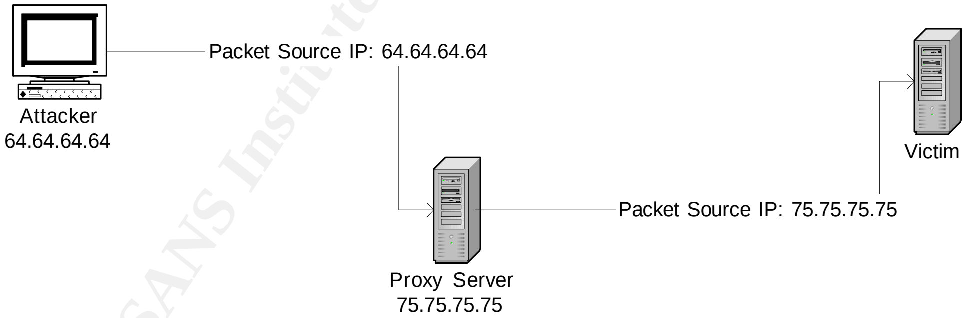
Log format
The Snort output shows you the name of the rule that has been violated, the classification of the alert and the priority. Next is the date and time stamp followed by the source IP address and source port. Next is the destination IP address and destination port. After that more detailed information about the packet including flags, TTL, etc.

3. Probability the source address was spoofed:

The chance of this IP address being spoofed is close to zero percent. This is an attacker scanning for open proxies, probably looking for anonymous proxies that they can direct their traffic through. The attacker wants to know the port is open so they need to be able to see the response from the destination IP address. The attacker could attempt an Nmap idle scan that would allow them to scan anonymously but that is a rare occurrence, so the chances are good that is this is not a spoofed IP address.

4. Description of attack:

This detect shows someone attempting to discover an open port that is usually reserved for proxies. Proxies allow other users to direct their traffic through them. Most likely the attacker is looking for anonymous proxies because anonymous proxies allow outside users to direct their traffic through them and then make the traffic appear as if it came from the proxy. Below is an example diagram.



In this example, if the “Attacker (64.64.64.64)” finds an anonymous open proxy running on the “Proxy Server (75.75.75.75)”, the “Attacker” can pass their traffic through the “Proxy Server”. Here the “Attacker” wants to attack the “Victim”. The “Attacker” launches an attack directed at the “Victim” but it passes its traffic through the anonymous “Proxy” first. This way the “Victim” sees the “Proxy” server as the one attacking it. Attackers can discover many different proxies and direct their traffic through many of them just to reach a single victim. This means the attacker could direct the traffic through 10 different proxies before reaching

the actual victim. This can make it very difficult to track down an attacker, especially when the proxy server resides in another country.
An interesting twist on this scan is that the attacker is using a source port of zero. I am not sure why the attacker has chosen to do this accept for the fact that either their tool does it by default or they feel they can bypass some perimeter defenses by using a source port of zero.

5. Attack mechanism:

This type of reconnaissance attack is a stimulus looking for a response. The attacker sends a SYN packet to a destination IP address and a destination port. If the attacker receives a SYN ACK from the target then they know that the port is open. If they receive a RST then they know the port is closed. Other possibilities are that they may receive no response because a perimeter device blocks it or the perimeter device may send the RST. Either way, the attacker is looking for the SYN ACK of an open port.
The attacker is looking for a proxy service. There have been vulnerabilities in the past with different proxy servers but this is most likely just an attempt to discover open proxies.

6. Correlations

While searching through the tcpdump files for the source IP address, many other alerts appeared as well and we can see that this IP address was scanning a number of servers for different ports.

```
10:57:37.555389 67.234.73.55.0 > MY.NET.25.78.8080: S [tcp sum ok] 11429793:11429793(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937       E..(.L@.q...C.I7
0x0010   c0a8 194e 0000 1f90 00ae 67a1 0000 0000        .l(N......g.....
0x0020   5002 0200 a127 0000 0000 0000 0000            P....'........
--
11:02:34.317802 67.234.73.55.0 > MY.NET.25.197.3128: S [tcp sum ok] 11438354:11438354(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937       E..(.L@.q..1C.I7
0x0010   c0a8 19c5 0000 0c38 00ae 8912 0000 0000        .l(....8........
0x0020   5002 0200 9297 0000 0000 0000 0000            P............
--
11:03:40.312555 67.234.73.55.0 > MY.NET.25.187.1080: S [tcp sum ok] 11464972:11464972(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937       E..(.L@.q..;C.I7
0x0010   c0a8 19bb 0000 0438 00ae f10c 0000 0000        .l(....8........
0x0020   5002 0200 32a7 0000 0000 0000 0000            P...2.........
--
11:09:54.849142 67.234.73.55.0 > MY.NET.25.0.3128: S [tcp sum ok] 11501847:11501847(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937       E..(.L@.q...C.I7
0x0010   c0a8 1900 0000 0c38 00af 8117 0000 0000        .l(....8........
0x0020   5002 0200 9b56 0000 0000 0000 0000            P....V........
--
```

```
11:12:06.667377 67.234.73.55.0 > MY.NET.25.157.1080: S [tcp sum ok] 11513983:11513983(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937        E..(.L@.q..YC.I7
0x0010   c0a8 199d 0000 0438 00af b07f 0000 0000         .l(....8........
0x0020   5002 0200 7351 0000 0000 0000 0000             P...sQ........
--
11:14:48.884132 67.234.73.55.0 > MY.NET.25.254.8080: S [tcp sum ok] 11511158:11511158(0)
win 512 (DF) (ttl 113, id 844, len 40)
0x0000   4500 0028 034c 4000 7106 ffff 43ea 4937        E..(.L@.q...C.I7
0x0010   c0a8 19fe 0000 1f90 00af a576 0000 0000        .l(........v....
0x0020   5002 0200 62a1 0000 0000 0000 0000             P...b.........
```

The ports being scanned for above are port 8080, 3128 and 1080. 8080 is a known proxy port, 3128 is a SQUID proxy port and 1080 is a SOCKS proxy port. This attacker is scanning many different IP addresses looking for different proxy services.
The use of zero as a source port was also noted by Vjay Larosa (http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00196.html) and Rob Bison (http://www.giac.org/practical/Rob_Bison.doc), but they give no theories on this port being used and I could find no other information on any reason for this source port.

7. Evidence of active targeting

This is probably not a targeted scan of my IP address space but is targeted for proxy servers. It is actively targeting proxy services and appears to be randomly hitting IP addresses. I believe it to be random because many of the IP addresses in the above tcpdump are not active and have nothing answering on those IP addresses.

8. Severity

Criticality: 1
It is scanning for proxy services on many IP addresses that have no machines bound to them. The one that does have a machine at that IP address is a web server for a new team in my company that is still in the test phase.

Lethality: 1
This is purely recon and will probably not have any adverse affects on any of machines unless there is very poorly written network application listening on one of those ports that a SYN scan would shutdown. If the attacker did find an open proxy, there could be legal issues later on if they used my network to stage an attack.

System Countermeasures: 4
To the best of my knowledge that system does not run a proxy server; however, since I don’t control it I would not be prepared to say that no proxy server runs on it.

Network Countermeasures: 5
The firewall is not configured to allow any of those ports inbound. We do have an internal proxy server but it is not reachable by outside machines.

Total Severity = (1+1)-(4+5) = -7

9. Defensive recommendation

The current defenses that are in place appear to be sufficient to handle this attack. However, there could be more proactive actions taken. For example, scanning the company's network (with proper permission) so that I know what services are running or if there are new servers that have cropped up on the network. Written policies explaining proxies and how the organization wants them setup would give the ability to stop anyone from attempting to setup an unauthorized proxy on the network.

10. Multiple Choice Questions

Proxies commonly run on all these ports except:

A) 7258
B) 1080
C) 8080
D) 3128
Answer: A

Part 3 – Analyze This

Executive Summary:

The report that follows was written after five consecutive days worth of data were reviewed. Three types of files were analyzed: alert logs, scan logs and out of spec logs. This report starts with listing the files that were analyzed and then moves into general recommendations about the network that were concluded after reviewing all of the information. Following that comes a more detailed report breaking down specific detects, top talkers and a more thorough analysis. The recommendations that follow each detect are aimed more at a technical level then at a management level and the recommendations are specific to that detect. At the end of the report, a description of the process used to analyze the data is given so that it can be seen how information was gathered and broken down.

List of files:

alert.040316
alert.040317
alert.040318
alert.040319
alert.040320
oos_report_040312
oos_report_040313
oos_report_040314
oos_report_040315
oos_report_040316
scans.040316
scans.040317
scans.040318
scans.040319
scans.040320

The oos_report files are different dates because the data inside them matched with the dates of the other files. Example, oos_report_040312 had data inside it that showed logs with time stamps that matched up with alert 040316.

Defense recommendations:

After analyzing the data that has been presented, it is apparent that certain security mechanisms should be utilized to offer better security on the network. First, Snort should be updated to a new version. New versions of Snort fix security issues of older versions and offer a more stable feature set. Also new analysis tools have been created that only work on the log format used by newer versions of Snort.

Next, access control lists should be utilized on perimeter firewalls or routers. Rules should be set up to limit the machines and ports that servers can communicate over. Currently it is difficult to tell what type of controls are being utilized by firewalls because no copy of the Snort rule set was available to aid in analyzing the data.

Anti-virus products need to be installed on machines and kept updated. There were signs of infections on multiple machines and current anti-virus could clean those machines infected and help prevent future infections. Host-based intrusion detection systems or integrity checkers are recommended for critical systems as well. This would allow critical systems to be more closely monitored.

List of detects:

Below are the 10 alerts that generated the most traffic over the five days worth of data.

MY.NET.30.4 activity	29883
MY.NET.30.3 activity	10939
SMB Name Wildcard	3494
EXPLOIT x86 NOOP	3243
Null scan!	1702
High port 65535 tcp - possible Red Worm - traffic	1306
NMAP TCP ping!	697
High port 65535 udp - possible Red Worm - traffic	526
Possible trojan server activity	418
Incomplete Packet Fragments Discarded	250

Detect #1 and #2 – MY.NET.30.4 activity and MY.NET.30.3 activity

Snort rule that triggers the alert:

These are custom rules written for Snort that monitors all inbound traffic to MY.NET.30.4 and MY.NET.30.3.

Snort alert:

```
[**] MY.NET.30.4 activity [**] 68.55.51.87:1975 -> MY.NET.30.4:51443
[**] MY.NET.30.4 activity [**] 68.55.51.87:1975 -> MY.NET.30.4:51443
[**] MY.NET.30.4 activity [**] 68.55.51.87:1975 -> MY.NET.30.4:51443
[**] MY.NET.30.4 activity [**] 68.55.51.87:1975 -> MY.NET.30.4:51443
```

These two detects have been grouped together because the rule is the almost the same for both. There are two custom rules that monitor inbound traffic to MY.NET.30.4 or MY.NET.30.3. Analyzing the log files show that these two machines receive numerous alerts from many external IP addresses. These machines are probably open to the Internet with no form of protection in front of them. The ports that receive traffic make it apparent that these are Netware machines running a web server on port 80, NCP service on port 524 and MY.NET.30.4 has port 51443 open. Internet searching for TCP port 51443 showed that Netware uses this port to run Apache web server over SSL for administration utilities and the iFolder application.

Many of the external hosts connected to all three ports – 80, 524 and 51443. There is no data in the oos_reports files or the scans files to show that hosts were actively scanning for 524 or 51443. This could mean that the external sources were redirected to port 51443 from 80 and then checked for port 524 after knowing that it was a Netware box.

No outbound traffic was seen from MY.NET.30.4 or MY.NET.30.3 because that must be part of the rule in Snort; the rule must just cover inbound traffic. Looking at all the connections to port 51443 and seeing that the source port stays the same from the outside connection, it is most likely an established connection has occurred and that outside sources are able to gain access to this site. This might be a concern seeing that 18 different IP addresses connected to port 51443 and 289 connected to MY.NET.30.4 on any ports.

Recommendations:

Alter the Snort rule to search for more specific items. Logging all incoming traffic sets off numerous alerts and makes it difficult to notice real threats. Outbound traffic should also be monitored because that is one of the best ways to track an attack. If only inbound traffic is logged, then there is no way to see the response and to detect if attacks were successful. If only certain machines should be able to communicate with MY.NET.30.3 and MY.NET.30.4 then rules should be placed on the firewalls or routers to limit traffic to those machines.

Detect #3 – SMB Name Wildcard

Snort rule that triggers the alert:

```
alert udp any any -> $HOME_NET 137 (msg:”SMB Name Wildcard";
content:"CKAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA|0000|";)
```

Snort alert:

```
03/20-22:50:50.777501  [**] SMB Name Wildcard [**] MY.NET.11.7:137 -> 169.254.25.129:137
```

This is usually very normal traffic that is generated by Windows machines during normal Internet use. Windows machines attempt to talk to other computers using the SMB protocol and it is common for a wildcard character to be used during this communication. Reviewing all of the alerts shows that all of them were triggered on outbound connection attempts, which supports that Window machines are attempting to communicate over SMB to other machines. However, it is something that should be monitored because looking through the scan files, there are many outside machines actively scanning for Windows ports including 135, 137 and 445. If these ports are found open, hackers will attempt to gain access through them and file-sharing should not need to take place outside of the network. It could be a good sign that all of these alerts show outbound traffic meaning that inbound traffic could already be filtered. However, it could also mean that the Snort rule is only picking up outbound traffic as well. Upon further analysis, there are two machines that draw attention to themselves in the files. MY.NET.150.198 and MY.NET.150.44 are the only two IP addresses in all the alerts that have a source port that is not 137. If two machines are talking SMB, then both source port and destination port will be 137. This alert has been analyzed in many other GCIA papers and Pete Storm (http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf [46]) discusses in his analysis that a source port other then 137 could be generated by file-sharing programs and IRC programs. Searching for alerts on just these IP addresses shows some interesting trends.

MY.NET.150.198

```
03/16-01:12:55.244957  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
210.136.90.146:137
03/16-01:13:55.059155  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
210.136.90.146:137
03/16-01:35:40.251804  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 -> 61.178.21.95:137
03/16-02:23:50.737821  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
80.35.227.239:137
03/16-02:24:16.113420  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
80.35.227.239:137
03/16-02:37:22.783021  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
24.195.127.101:137
03/16-02:50:01.156038  [**] SMB Name Wildcard [**] MY.NET.150.198:1074 ->
220.108.33.244:137
03/16-03:36:31.661427  [**] SMB Name Wildcard [**] MY.NET.150.198:1071 ->
61.207.196.101:137
```

MY.NET.150.44

```
03/16-16:37:40.703491  [**] SMB Name Wildcard [**] MY.NET.150.44:1062 -> 81.56.198.138:137
03/16-17:27:05.172912  [**] SMB Name Wildcard [**] MY.NET.150.44:1059 -> 169.233.37.91:137
03/16-17:28:21.319153  [**] SMB Name Wildcard [**] MY.NET.150.44:1059 -> 169.233.37.91:137
03/16-17:39:26.143413  [**] SMB Name Wildcard [**] MY.NET.150.44:1059 ->
194.149.152.241:137
```

Both of these machines are showing signs of scanning. MY.NET.150.198 appears to be checking many different machines on a fairly regular basis. This could be a sign of file-sharing programs; however, it happens consistently throughout the day and night. This could be that it is a file-sharing application that is turned on all the time or a worm that is attempting to spread itself. MY.NET.150.44 exhibits some of the same type of behavior as 150.198 except that searching the logs shows that it has had exploits launched towards it on port 80. Another point of interest is that it will connect to an outside IP address using a source port of 137 after it attempted a connection with a different source port. The log below makes this clearer.

```
03/16-09:00:28.442280  [**] SMB Name Wildcard [**] MY.NET.150.44:1062 -> 67.71.54.239:137
03/16-08:59:10.862787  [**] SMB Name Wildcard [**] MY.NET.150.44:137 -> 67.71.54.239:137
```

```
03/16-11:54:38.841210  [**] SMB Name Wildcard [**] MY.NET.150.44:1062 ->
212.182.165.88:137
03/16-11:54:39.610663  [**] SMB Name Wildcard [**] MY.NET.150.44:137 -> 212.182.165.88:137
```

This makes it appear as if it is scanning for port 137 and once found, it opens a connection to it.

Recommendations:
Check the Snort rule and make sure that it monitors for inbound connections since inbound would be of a greater concern. If the firewalls or routers are not already set up to be blocking Windows ports, then they should block TCP and UDP port 135, 137, 138, 139 and 445. These ports should rarely need to be opened to the outside world and are targeted by many viruses and hackers. Anti-virus also needs to be installed on all machines and kept updated. A policy should be written about peer-to-peer file sharing and if it is not allowed then it should be blocked. MY.NET.150.198 and MY.NET.150.44 should be investigated further.

Detect #4 – EXPLOIT x86 NOOP

Snort rule that triggers the alert:

```
alert tcp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP"; content: "|90
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90|"; flags: A+;
reference:arachnids,181;)

alert udp $EXTERNAL_NET any -> $HOME_NET any (msg:"EXPLOIT x86 NOOP";
content:"|9090 9090 9090 9090 9090 9090 9090 9090|"; reference:arachnids,181;)
```

Snort alert:

```
03/16-00:05:31.046732  [**] EXPLOIT x86 NOOP [**] 130.160.155.41:3527 ->
MY.NET.82.113:1025
```

This rule is looking for what is called a “NOOP Slide”, which is very common in buffer overflows being exploited. A buffer overflow is caused when somewhere in the code of a program; a variable fails to check how much input it is receiving. When the vulnerability is exploited, a large amount of data is pushed into the buffer in an attempt to overflow it. In the data that is being pushed into the buffer, is code that the hacker wants the system to execute. The hacker has get to their code onto the stack of the system and then has to be able to reference it in memory. It can be difficult to know the exact memory location of their code, so they will sometimes pad the buffer with NOOPs (no operation machine code and instruction 0x90). That way if they hit a memory address anywhere in the “NOOP”, the pointer will slide down them until it reaches their code. With this in mind, the above rule checks for numerous NOOPs because it could be a sign that a buffer overflow exploit has been attempted.
This rule is normally turned off in Snort when it is first installed because it can create many false positives. Looking through the alerts file at least one entry shows something interesting.

```
03/18-06:00:42.687070  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:4920 -> MY.NET.5.46:80
03/18-06:01:04.813689  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:3081 -> MY.NET.5.46:80
03/18-06:01:11.993250  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:3639 -> MY.NET.5.76:80
03/18-06:01:13.346771  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:1845 -> MY.NET.11.2:80
03/18-06:01:23.844285  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:3736 -> MY.NET.5.46:80
03/18-06:01:27.736803  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:2117 -> MY.NET.5.76:80
03/18-06:01:29.662363  [**] EXPLOIT x86 NOOP [**] 218.152.49.141:2567 -> MY.NET.11.8:80
```

Another host, 61.129.45.60 appears to do the same thing as shown below.

```
03/20-05:43:23.917864  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:4948 -> MY.NET.6.15:80
03/20-05:43:23.934111  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:4954 -> MY.NET.6.14:80
03/20-05:43:24.438381  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:1084 -> MY.NET.5.20:80
03/20-05:43:24.563685  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:1135 -> MY.NET.5.25:80
03/20-05:43:24.595952  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:1138 -> MY.NET.5.26:80
03/20-05:43:26.657636  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:1273 -> MY.NET.5.34:80
03/20-05:43:26.938375  [**] EXPLOIT x86 NOOP [**] 61.129.45.60:1310 -> MY.NET.5.44:80
```

These show one host setting off alerts while going to different hosts, all on port 80. It could be someone trying out an exploit against web servers. The machines should be checked for possible signs of exploitation and should be monitored for outbound connections.

Further examinations of the logs reveals numerous port scans coming from 218.152.49.141 which could have been a recon attack followed by this launch of exploit code. The scan files reveal this same host as also scanning for port 3389 which is normally reserved for Windows Terminal Services. Looking this host up through a whois shows that it is registered to a Korean Telecom raising more suspicion about this traffic. All of the factors that it is a Korean IP address, scanning for port 80 and 3389 and launching attacks make it clear that there is malicious intent in this traffic.

The alert files also reveal that 61.129.45.60 performed numerous port scans against the network. The scan files show this host scanning numerous times looking for port 80. This once again shows the signs of malicious intent and should be monitored. This IP address is registered to a company in China. Foreign IP addresses raise the suspicion level on attacks because would-be attackers know that coming from overseas makes it difficult to track them. It is difficult to get cooperation with locations in other countries when trying to track an attacker. Below is an example from the alerts file showing a port scan.

```
03/20-05:55:25.841426  [**] spp_portscan: portscan status from 61.129.45.60: 25 connections
across 25 hosts: TCP(25), UDP(0) [**]
```

Recommendations:

Repeat offenders should be blocked at the perimeter. Blocking 218.152.49.141 at the perimeter will stop this attacker from any further activity. However, this attackers ISP is probably using DHCP to assign addresses to its customers and this would only block this attacker until they get a different IP address. To block or not to block can be argued either way. Blocking the attacker may let them know you noticed this traffic and they might think twice about returning, or it could entice them to come back more often. To keep blocking single IP addresses could become a tedious task.

Detect #5 – Null Scan

Snort rule that triggers the alert:

```
alert tcp any any -> 192.168.160.0/24 any (msg:"NULL Scan"; flags: 0;)
```

Snort alert:

```
03/16-00:15:29.853849  [**] Null scan! [**] 68.122.128.1:15131 -> MY.NET.12.4:110
```

A null scan is when a packet has no flags set. For example, there is no SYN flag or RST flag, etc. This could be a sign of someone attempting to discover open ports and is hoping that a Null scan will be able to bypass any perimeter protection on the network. In the past, some firewalls have had vulnerabilities or misconfigurations that would allow a Null packet to get passed through to the internal network.
Looking through all the alert files, one instance is a somewhat interesting because the source IP, 61.48.11.22, scans using a source port of ‘0’ and a destination port of ‘0’. Review the scan files also shows this host attempting a number of other interesting scan attempts

Snort alert example:

```
03/19-12:15:35.941881  [**] Null scan! [**] 61.48.11.22:0 -> MY.NET.153.76:0
```

Snort scan example:

```
Mar 19 12:15:47 61.48.11.22:64084 -> 130.85.153.76:49758 INVALIDACK 1*UAP*SF
RESERVEDBITS
```

The scan that is using a source port of zero and a destination port of zero could be an attempt to identify the operating system of MY.NET.153.76. Different operating systems will respond to Null scans and scans directed at port zero differently. It could also be an attempt to see if port 0 can get past perimeter devices and can discover if a machine is alive or not.

Recommendations:
This type of traffic is usually recon traffic either by attackers or just curious Internet users playing with tools. These types of alerts are most useful in correlating an attack. Many times when researching one alert, you might be able to find the first time the attacker mapped your network by looking at different scan attempts. This alert is not a high priority but aids in correlating other events. Make sure that port 0 does not bypass your perimeter security devices.

Detect #6 - High port 65535 tcp - possible Red Worm – traffic

Snort rule to trigger the alert:
I couldn’t find it in the default set of rules so it appears to be a custom rule. It appears to be looking for traffic on TCP port 65535.

Snort alert:

```
03/20-05:47:43.628515  [**] High port 65535 tcp - possible Red Worm - traffic [**]
220.220.220.1:65535 -> MY.NET.24.47:2304
```

According to description written by Matt Fearnow and William Stearns (http://www.sans.org/y2k/adore.htm) Adore was originally called the Red Worm and it scans the Internet looking for machines with vulnerable versions of LPRng, rpc-statd, wu-ftpd and BIND on Linux machines. Once a machine is infected, it will open up a listener on port 65535. The listening port and protocol could be changed in the code as well so you could see Adore listening on any port using TCP or UDP. This rule is trigged many times and many are probably false positives. I am intrigued by the amount of systems that are communicating between port 65535 and port 25. I can find no information on the Internet that shows this type of activity and browsing through the alert file there are so many of them that it is appearing as normal. One that did stick out of the alert files was this:

```
03/16-22:35:33.543265  [**] High port 65535 tcp - possible Red Worm - traffic [**]
220.247.201.20:25 -> MY.NET.25.10:65535
03/16-22:35:33.543749  [**] High port 65535 tcp - possible Red Worm - traffic [**]
MY.NET.25.10:65535 -> 220.247.201.20:25
```

What was intriguing about this is the amount of traffic between the two hosts. 151 alerts where generated by these two hosts talking back and forth. 220.247.201.20 was further researched in external sources below and was found to belong to an organization in another country and the organization appears to no longer exist. Scanning the logs also shows MY.NET.25.10 communicating to 64.8.48.7 on port 25 as well. This type of traffic has been noted in other practicals as well including Marshall Heilman's (http://www.giac.org/practical/GCIA/Marshall_Heilman_GCIA.doc.pdf [52]). He noted the same traffic that we see here with a port of 25 communicating to a port 65535 over different hosts. He notes that it is suspicious to see the port 65535 being used multiple times by multiple addresses. The machines using a source port of 25 could be attempting to bypass firewall rules by pretending to be SMTP traffic.

Recommendations:

MY.NET.25.10 should be investigated further to check for signs of infection. The Snort rule should also be modified to look for specific signs of infection by the adore worm to cut down on the number of false positives.

Detect #7 - NMAP TCP ping!

Snort rule to trigger the alert:

```
alert tcp any any -> any any (flags: A; ack: 0; msg:"NMAP TCP ping!";)
```

Snort alert:

```
03/16-01:23:20.921085  [**] NMAP TCP ping! [**] 63.211.17.228:80 -> MY.NET.1.3:53
03/16-01:23:20.921097  [**] NMAP TCP ping! [**] 63.211.17.228:53 -> MY.NET.1.3:53
03/16-01:32:24.081176  [**] NMAP TCP ping! [**] 61.31.32.248:57531 -> MY.NET.24.44:80
03/16-01:32:24.105666  [**] NMAP TCP ping! [**] 211.21.160.70:85 -> MY.NET.24.44:80
03/16-01:32:24.116518  [**] NMAP TCP ping! [**] 203.75.25.62:84 -> MY.NET.24.44:80
03/16-01:32:24.119588  [**] NMAP TCP ping! [**] 61.222.103.122:64105 -> MY.NET.24.44:80
03/16-01:32:24.148389  [**] NMAP TCP ping! [**] 61.222.103.141:58677 -> MY.NET.24.44:80
```

This rule is triggered when Snort sees a packet with the ACK flag set and the acknowledgement number is zero. This can be a sign of the NMAP TCP ping which is used to by hackers to discover machines that do not respond to ICMP traffic or ones that are behind devices that block ICMP traffic. It is also a default ping to port 80 used if Nmap is run with pings turned on. This was noted in part 1 of this paper. `The hacker will use this tool for reconnaissance looking for machines that are alive. They will use a TCP ping along with a well-known port that has a good chance of being let through a firewall. By default, Nmap will perform an ICMP ping and a TCP ping using a packet with the ACK flag set and aimed at destination port 80 to determine if a machine is up. In the example above, there is a nice variety of tricks shown that a hacker can use. In the first two examples, they have tried a TCP Ping aimed at TCP port 53 and set their own source port to 80 once and 53 the next time. Pinging for port 53 can work because many firewalls allow port 53 traffic through because it is commonly used for DNS. The source port was changed because the chance was that the first attempt didn’t work and the attacker is looking to see if the firewall allows machines to talk to 53 when the source port is 53. The other common port scanned for is port 80 because almost all firewalls have to let port 80 through to some machines to allow for web traffic.

These alerts are probably just recon but can be useful in correlating other attacks are seen. If alerts are picked up and the offending IP address is searched for, you may see an NMAP TCP Ping alert showing that the attacker did indeed do some recon before launching the rest of their attack. This helps identify targeted attacks as well.

Recommendations:

Firewalls should be configured to not allow packets through with an acknowledgement number of 0 because this not a possible acknowledgement number and is a sign of crafted packets. Otherwise this traffic is just recon looking for live machines and is not a high priority alert. A stateful firewall should be used as well and that way packets that have the ACK flag set but have no previous communication from an internal device would be blocked. This type of alert is useful in correlating attack data to see if it goes with any other alerts. If this type of traffic is seen prior to other malicious traffic, then you can see when the attacker first struck and can see that it is possibly a focused attack.

Detect #8 - High port 65535 udp - possible Red Worm - traffic

Snort rule:

Custom rule that appears to be looking for UDP traffic with either a destination or source port of 65535

Snort alert:

```
03/16-09:23:10.787719  [**] High port 65535 udp - possible Red Worm - traffic [**]
65.25.233.39:65535 -> MY.NET.70.164:4672
03/16-09:23:10.789007  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.70.164:4672 -> 65.25.233.39:65535
03/16-10:20:42.818056  [**] High port 65535 udp - possible Red Worm - traffic [**]
66.250.188.23:65535 -> MY.NET.66.29:53474
03/16-12:11:15.036749  [**] High port 65535 udp - possible Red Worm - traffic [**]
64.83.35.113:65535 -> MY.NET.1.4:53
03/16-14:45:40.883116  [**] High port 65535 udp - possible Red Worm - traffic [**]
66.118.159.153:65535 -> MY.NET.84.216:6257
03/16-15:21:54.215629  [**] High port 65535 udp - possible Red Worm - traffic [**]
66.118.159.153:65535 -> MY.NET.84.216:6257
```

This detect has a lot in common with detect #6. The only difference in this rule is that Snort is now looking for UDP traffic. The Adore/Red worm could be altered to talk on either protocol so it is a good measure to watch for both protocols. The interesting thing about UDP is that it can aid in finding infected machines because if you see two machines talking back and forth over UDP on port 65535, that would show a pretty good sign of infection. TCP requires a three-way handshake to make a connection so if a machine sends a SYN packet to port 65535 on a machine then that the machine will respond back with a RST or SYN ACK packet still using port 65535. An example is below:

```
19:13:41.926168 192.168.0.6.56581 > 192.168.0.3.65535: S [tcp sum ok]
3607570151:3607570151(0) win 65535 <mss 1460,nop,wscale 1,nop,nop,timestamp 171588346
0> (DF) [tos 0x10]  (ttl 64, id 7153, len 60)
19:13:41.926309 192.168.0.3.65535 > 192.168.0.6.56581: R [tcp sum ok] 0:0(0) ack 3607570152
win 0 (ttl 128, id 4159, len 40)
```

Here you can see the RST packet sent using port 65535. This would set off the Snort alert twice in TCP rule. However, UDP is connectionless and the destination machine doesn't have to talk back. If you send a UDP packet to a machine that has a closed port, you will only see and ICMP message back. So if you see a conversation being carried on, it could be a sign of infection. Below are some alerts of interest:

```
03/18-03:12:22.766298  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:12:23.140682  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:12:43.540405  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:12:45.562202  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:31:30.894434  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:12:50.162750  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:12:53.307350  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
```

```
03/18-03:31:38.274926  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:12:54.780699  [**] High port 65535 udp - possible Red Worm - traffic [**]
69.140.137.209:65535 -> MY.NET.6.62:65535
03/18-03:31:38.608669  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:31:39.200818  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:31:39.643689  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:31:39.716038  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
03/18-03:31:40.110797  [**] High port 65535 udp - possible Red Worm - traffic [**]
MY.NET.6.62:65535 -> 69.140.137.209:65535
```

Notice that both machines are talking to each other over UDP port 65535. This could be a very good sign of infection. It is also interesting to note that these alerts were set off at 3:31 AM. No other alerts where generated by these hosts. Other papers have put less emphasis on this alert. Some papers recommended turning this rule off because Red Worm is not known to talk on UDP. I question this because the Red Worm can be altered to talk over other protocols.

Recommendations:
This Snort rule should be altered to see if it can look more specifically for Red Worm traffic with less false positives. This alert generates a lot of noise in the logs and much of it appears to be just network noise. MY.NET.6.62 should be analyzed further because both source port and destination port of 65535 raises suspicion.

Detect #9 - Possible Trojan server activity

Snort rule:
Custom rule that is looking for port 27374 as either a destination port or source port

Snort alert:

```
03/20-04:24:43.300203  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
03/20-04:24:43.379333  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
03/20-04:24:43.410375  [**] Possible trojan server activity [**] 66.90.79.46:27374 ->
MY.NET.97.74:9277
03/20-04:24:43.764712  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
03/20-04:24:43.794755  [**] Possible trojan server activity [**] 66.90.79.46:27374 ->
MY.NET.97.74:9277
03/20-04:24:43.900931  [**] Possible trojan server activity [**] 66.90.79.46:27374 ->
MY.NET.97.74:9277
03/20-04:24:44.372210  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
```

```
03/20-11:17:19.697062  [**] Possible trojan server activity [**] 193.130.33.3:27374 ->
MY.NET.24.44:80
```

Port 27374 is a well-known Trojan port used by the backdoor program called SubSeven. SubSeven is well-described by Georg Wagner and his analysis can be found at http://www.sans.org/y2k/subseven.htm. SubSeven is a Trojan for the Windows platform that uses a client/server architecture. The server is installed on the victim's machine and the hacker uses the client to connect to the server. When an infected machine is online, it can be setup to notify the attacker via IRC, ICQ or email that the machine is up and give the attacker the infected machine's IP address. These are also good ways to look for infected machines. These alerts can cause a lot of false positives since a machine could be using port 27374 legitimately. An example is a client connecting to a web server on port 80, may open up port 27374 on its local machine. An example of this is:

```
03/16-05:45:55.426867  [**] Possible trojan server activity [**] 194.167.235.251:27374 ->
MY.NET.24.34:80
03/16-05:45:55.427120  [**] Possible trojan server activity [**] MY.NET.24.34:80 ->
194.167.235.251:27374
03/16-05:45:55.427353  [**] Possible trojan server activity [**] MY.NET.24.34:80 ->
194.167.235.251:27374
03/16-05:45:55.427587  [**] Possible trojan server activity [**] MY.NET.24.34:80 ->
194.167.235.251:27374
```

There is a chance that this could be Trojan activity, but it is most likely just web traffic. Searching the files for other references of these IP addresses turned up nothing suspicious. Some suspicious traffic was shown in our original example:

```
03/20-04:24:42.899291  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
03/20-04:24:42.930221  [**] Possible trojan server activity [**] 66.90.79.46:27374 ->
MY.NET.97.74:9277
03/20-04:24:42.939154  [**] Possible trojan server activity [**] 66.90.79.46:27374 ->
MY.NET.97.74:9277
03/20-04:24:43.300203  [**] Possible trojan server activity [**] MY.NET.97.74:9277 ->
66.90.79.46:27374
```

Neither of these IP addresses shows up in any of the other files; however, this is suspicious because of the ports that the communication is taking place over. Port 9277 is a known listening port for a Windows Trojan called BackGate making it possible that MY.NET.97.74 is infected with BackGate and 66.90.79.46 is controlling it.

Another interesting example:

```
03/18-01:45:15.023582  [**] Possible trojan server activity [**] 67.114.251.118:27374 ->
MY.NET.24.47:4883
03/18-01:45:15.023664  [**] Possible trojan server activity [**] MY.NET.24.47:4883 ->
67.114.251.118:27374
```

This is interesting here because of the ports that are talking. No well-known service runs on port 4883 and checking the other files shows a lot of activity aimed at MY.NET.24.47 including FTP passwd attempt alerts, NULL Scan alerts and possible Red Worm alerts. It is possible that this machine is being used to control other machines. Here are some of the alerts that were mentioned.

```
03/18-06:03:11.898764  [**] FTP passwd attempt [**] 163.28.4.1:63759 -> MY.NET.24.47:21
03/20-05:26:13.273111  [**] High port 65535 tcp - possible Red Worm - traffic [**]
220.220.220.1:65535 -> MY.NET.24.47:2157
```

In these examples, MY.NET.24.47 is talking to machines with destination ports that match a Trojan and a machine is logging into it through FTP. A large number of IP addresses are connecting to this machine on FTP which could signal that it has been compromised and is being used for storing files. The times of all the above alerts are interesting because of the early hours that these events occur – 1 AM, 6 AM and 5:26 AM.

Recommendations:
Investigate the following machines: MY.NET.97.74 and MY.NET.24.47. These two machines both show suspicious traffic and more information needs to be gathered to make a more informed analysis. The Snort rule also should be modified to look for more specific items. Just alerting on source or destination ports can create a lot of false positives since any ephemeral port can be selected by an application for communication. The more false positives generated, the more chance a real attack might get through as the IDS administrator is loaded down looking at so many alerts.

Detect #10 - Incomplete Packet Fragments Discarded

Snort rule:
This is not a normal rule setting this off, it is actually a Snort preprocessor called defrag. A Snort preprocessor is "run before the detection engine is called, but after the packet has been decoded" (http://www.snort.org/docs/snort_manual/node17.html).

Snort alert:

```
03/20-19:26:38.698657  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.101.254:0
03/20-19:36:05.493780  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.21.194:0
03/20-19:46:20.242160  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.33.28:0
03/20-19:42:43.151063  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.112.60:0
03/20-20:03:33.852714  [**] Incomplete Packet Fragments Discarded [**] 151.196.43.67:0 ->
MY.NET.11.4:0
03/20-20:19:05.955552  [**] Incomplete Packet Fragments Discarded [**] 68.95.2.66:0 ->
MY.NET.69.226:0
```

```
03/20-20:37:42.330673  [**] Incomplete Packet Fragments Discarded [**] 151.196.43.67:0 ->
MY.NET.11.4:0
03/20-20:28:42.152677  [**] Incomplete Packet Fragments Discarded [**] 68.95.2.66:0 ->
MY.NET.69.226:0
```

There were a number of these alerts generated with many varying source IP addresses and destination IP addresses. There are a few source IPs that target multiple destination IP addresses as in the above example, we see 63.196.194.176 scans four different destination IP addresses. The other item of note is that all the source ports and destination ports are '0'. Searching the Internet looking for information pertaining to this alert, a posting by Marty Roesch, the developer of Snort, stated that this was an older defrag preprocessor alert and that it had "nasty failure modes" (http://www.mcabee.org/lists/snort-users/Nov-01/msg00820.html). In his post, he recommends upgrading to the new preprocessor called "frag2". Another post (http://www.geocrawler.com/archives/3/4890/2001/2/350/5151528/) on the same subject was made by Dragos Ruiu, another developer of Snort, giving more information on what causes this alert. This alert can be caused by transmission errors, broken stacks or fragmentation attacks. The first two causes would not be security concerns; however, fragmentation attacks would be a security concern because it could be attackers attempting to bypass firewalls and avoid IDS detection. Searching the log files for the IP address 63.196.194.176 shows that other alerts were generated by the same IP address.

```
03/20-19:33:30.098289  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.111.167:0
03/20-19:26:38.698657  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.101.254:0
03/20-19:36:05.493780  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.21.194:0
03/20-19:46:20.242160  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.33.28:0
03/20-19:42:43.151063  [**] Incomplete Packet Fragments Discarded [**] 63.196.194.176:0 ->
MY.NET.112.60:0
03/20-22:01:20.141295  [**] spp_portscan: PORTSCAN DETECTED from 63.196.194.176
(STEALTH) [**]
03/20-21:45:04.977203  [**] Null scan! [**] 63.196.194.176:0 -> MY.NET.153.198:0
03/20-22:01:24.947245  [**] spp_portscan: portscan status from 63.196.194.176: 1 connections
across 1 hosts: TCP(1), UDP(0) STEALTH [**]
03/20-22:01:30.520982  [**] spp_portscan: End of portscan from 63.196.194.176: TOTAL
time(0s) hosts(1) TCP(1) UDP(0) STEALTH [**]
Mar 20 21:45:04 63.196.194.176:0 -> 130.85.153.198:0 NULL ********
```

We can see from these alerts that the attacker did some other activity that appears to be reconnaissance work. It looks as though no malicious activity was generated but the attacker appears to be trying different methods to discover machines on the network. Doing searches on other IP addresses that set off this alert revealed no further information or alerts. Most of these alerts are probably false positives caused by poorly written applications or other network issues.

Recommendations:
Snort should be updated to use the frag2 preprocessor instead of the current defrag preprocessor. Since there were known issues with that preprocessor, much of this traffic could be false positives created by Snort. It is important to monitor for fragmentation since it is not normal traffic and it would most likely be a sign of malicious traffic.

Top Talkers:

The top talkers list was looked at on more than one level. In the first pass, the main criteria for looking for the top ten talkers was to mainly focus on the MY.NET addresses. The interest here is to see what machines on this network are talking the most and generating the most alerts. To find the most talkative a series of commands were run against the concatenated alerts file to find the most reoccurring IP address in the MY.NET address space. These numbers where than compared to the oos_reports and scans files looking for differences that could skew the results.
Based off the alerts file, the top talkers in MY.NET address space are as follows:

# of alerts	IP address
29864	MY.NET.30.4
10936	MY.NET.30.3
1094	MY.NET.11.7
971	MY.NET.153.76
688	MY.NET.24.47
557	MY.NET.75.13
370	MY.NET.1.3
364	MY.NET.150.198
274	MY.NET.153.80
251	MY.NET.84.216

Next I looked at the top ten alert generators. This took into account which two talking systems set off the most alerts. This was created using a Perl script called snort_stat.pl (http://packetstormsecurity.nl/sniffers/snort/snort_stat.pl). MY.NET.30.4 and MY.NET.30.3 accounted for a majority of these alerts so they were disregarded to make sure that their traffic did not skew the results.

# of alerts	From	To	Alert
1093	MY.NET.11.7	169.254.25.129	SMB Name Wildcard
963	61.48.11.22	MY.NET.153.76	Null scan!
325	MY.NET.24.47	220.220.220.1	Possible Red Worm (TCP)
243	202.43.239.178	MY.NET.153.80	Null scan!
184	220.220.220.1	MY.NET.24.47	Possible Red Worm (TCP)
132	64.152.70.68	MY.NET.1.3	NMAP TCP ping!
131	63.211.17.228	MY.NET.1.3	NMAP TCP ping!
120	69.140.137.209	MY.NET.6.62	Possible Red Worm (UDP)
106	68.122.128.1	MY.NET.12.4	Null scan!
100	MY.NET.25.10	220.247.201.20	Possible Red Worm (TCP)

Five selected external source addresses:

Below are five external IP addresses chosen to be searched for registration information. These five addresses were chosen for different reasons, but it is always a good practice to look up information on outside hosts that trigger interesting alerts because the registration information gathered can give more insight into certain alerts. All the information below comes from using Geektools whois at http://www.geektools.com/whois.php.

The IP address 218.152.49.141 was chosen because it was seen in one of the top ten detects, exploit NOOP alert, and was found to be performing port scanning on the network. The port scanning could have been reconnaissance to discover targets to launch its exploit against. Here is the information:

```
inetnum: 218.144.0.0 - 218.159.255.255
netname: KORNET
descr: KOREA TELECOM
descr: Network Management Center
country: KR
admin-c: DL248-AP
tech-c: GK40-AP
remarks: ******************************************
remarks: Allocated to KRNIC Member.
remarks: If you would like to find assignment
remarks: information in detail please refer to
remarks: the KRNIC Whois Database at:
remarks: http://whois.nic.or.kr/english/index.html
remarks: ******************************************
mnt-by: MNT-KRNIC-AP
mnt-lower: MNT-KRNIC-AP
changed: hostmaster@apnic.net 20010924
status: ALLOCATED PORTABLE
source: APNIC

person: Dong-Joo Lee
address: 128-9 Yeong-Dong Jongro-Ku Seoul
address: Network Management Center
country: KR
phone: +82-2-766-1407
fax-no: +82-2-766-6008
e-mail: ip@ns.kornet.net
nic-hdl: DL248-AP
mnt-by: MAINT-NEW
changed: hostmaster@nic.or.kr 20010425
source: APNIC

person: Gyung-Jun Kim
address: KORNET
address: 128-9, Yeong-Dong, Jongro-Ku
address: SEOUL
address: 110-763
country: KR
phone: +82-2-747-9213
fax-no: +82-2-3673-5452
```

```
e-mail: ip@ns.kornet.net
nic-hdl: GK40-AP
mnt-by: MNT-KRNIC-AP
changed: hostmaster@nic.or.kr 20010906
source: APNIC
```

This shows that this source IP is coming from Korea and it appears to be a Korean ISP because of the size of the net block they own and they are labeled as a telecom. There is no individual listing specific for this IP showing that it is owned by a business so it is probably a home user of this ISP. This makes it a possibility that other attacks could come from the same attacker but could be using different IP addresses if this particular ISP uses DHCP on their network.

The next IP picked is 61.129.45.60 and it was seen in the ‘Exploit NOOP’ alert section and was also performing port scans just as the host above. Here is there registration information:

```
inetnum: 61.129.45.48 - 61.129.45.83
netname: null
descr: null
country: CN
admin-c: WQ58-AP
tech-c: WL371-AP
mnt-by: MAINT-CHINANET-SH
changed: wanglin@shaidc.com 20040413
status: ASSIGNED NON-PORTABLE
source: APNIC

person: Wang Qing
address: 6F,380 Fushan Road,Shanghai 200122
country: CN
phone: +86-21-68761255-807
fax-no: +86-21-68761255-805
e-mail: wanglin@shaidc.com
nic-hdl: WQ58-AP
mnt-by: MAINT-CN-SHTELE-XINCHAN
changed: wanglin@shaidc.com 20021007
source: APNIC

person: Wang Lin
address: 6F,380 Fushan Road,Shanghai 200122
country: CN
phone: +86-21-68761255-807
fax-no: +86-21-68761255-805
e-mail: wanglin@shaidc.com
nic-hdl: WL371-AP
mnt-by: MAINT-CN-SHTELE-XINCHAN
changed: wanglin@shaidc.com 20021007
source: APNIC
```

In this example, a much smaller block is owned and it is coming for China as seen in the country code. It is interesting here that the netname and description are both ‘null’. Visiting their website shaidc.com makes it appear as though this is a hosting company as well. Even though the site is in a foreign language, there are some English words that give clues to this. For example, you can see the words “domain”, “email”, “host”, “website” and “promote” across the top of the page. This probably means this IP address belongs to a home user in China and that if this ISP is using DHCP to assign IP addresses, then multiple attacks could come from this same attacker using different IP addresses in the range owned by the ISP.

The IP 220.247.201.20 was chosen because it was seen in possible red worm alerts and traffic was seen going to and coming from MY.NET.25.10. Here is the information:

```
inetnum: 220.247.201.0 - 220.247.201.127
netname: SONIC-SLT-LK
country: LK
descr: Sonicnet Technologies (Pvt.) Ltd.
descr: 3rd Floor, Austraila Bldg., York Street, Colombo-01.
admin-c: MC662-AP
tech-c: MC662-AP
status: ASSIGNED NON-PORTABLE
changed: hostmaster@slt.lk 20030702
mnt-by: MNT-SLT-LK
source: APNIC

person: Muhunthan Canagey
nic-hdl: MC662-AP
e-mail: muhunthan@epsi.lk
address: 3rd Floor, Australia bldg, York Steet, Colombo 01
phone: +94-777-723310
fax-no: +94-1-372371
country: LK
changed: hostmaster@slt.lk 20030702
mnt-by: MNT-SLT-LK
source: APNIC
```

This source IP is coming from the country LK which is listed as being Ceylon (now Sri Lanka). From the registration, it appears to be a company called Sonicnet Technologies that owns this block. However, attempting to go to the websites listed, slt.lk and epsi.lk, reveals that neither of those sites are up so no more information can be gained by this. Searching the Internet for Sonicnet Technologies revealed no hits on this company. Going to port 80 of the IP 220.247.201.20 shows a page that just says “Under construction”.

Next IP address selected was 69.140.137.209 because it was seen in possible UDP Red Worm alerts and communicated back and forth with MY.NET.6.62. Here is its information:

```
Comcast Cable Communications, Inc. JUMPSTART-3 (NET-69-136-0-0-1)
69.136.0.0 - 69.143.255.255
Comcast Cable Communications, Inc DC15-NROCK1 (NET-69-140-0-0-1)
69.140.0.0 - 69.140.255.255
```

This IP address is part of Comcast Cable's network. Comcast is a cable company that offers high-speed cable Internet. This address most likely belongs to one of their home user subscribers and this IP address could change because of DHCP.

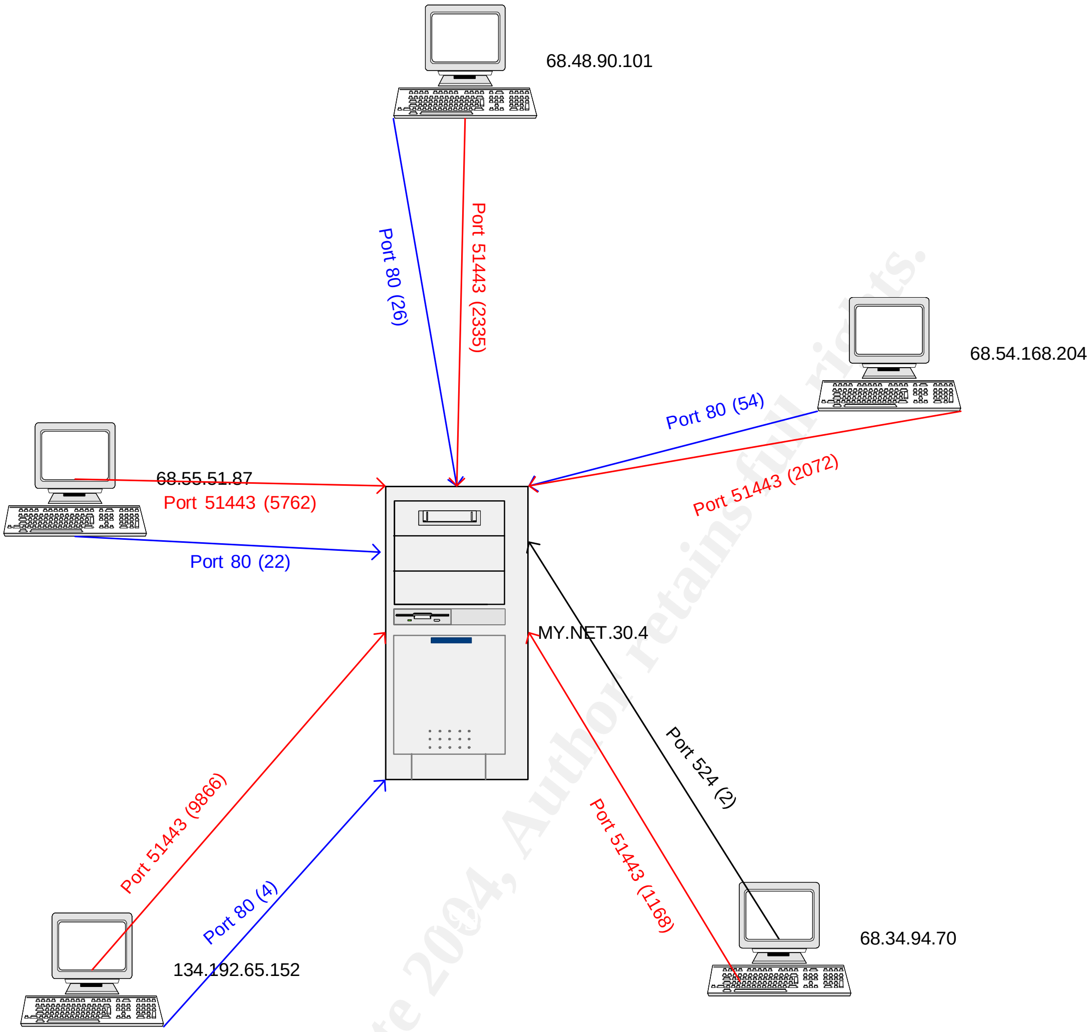
134.192.65.152 was selected because it is the outside IP address that caused the most alerts. Here is the information:

```
OrgName: University of Maryland at Baltimore
OrgID: UMAB-1
Address: 601 W. Lombard St.
City: Baltimore
StateProv: MD
PostalCode: 21201
Country: US

NetRange: 134.192.0.0 - 134.192.255.255
CIDR: 134.192.0.0/16
NetName: UMAB-NET
NetHandle: NET-134-192-0-0-1
Parent: NET-134-0-0-0-0
NetType: Direct Assignment
NameServer: NMS.UMARYLAND.EDU
NameServer: COMM1.UMARYLAND.EDU
Comment:
RegDate: 1988-06-15
Updated: 2000-11-09

TechHandle: FS178-ARIN
TechName: Smith, Frederick
TechPhone: +1-410-706-8337
TechEmail: fsmith@umaryland.edu
```

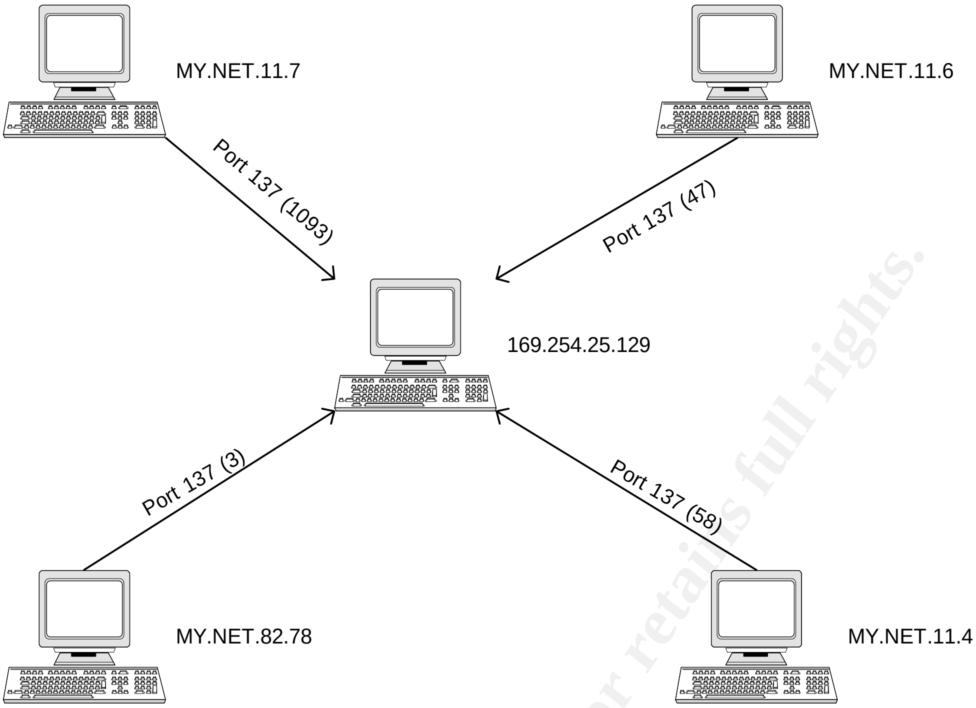
This IP address belongs to the University of Maryland. This IP caused 9950 alerts but all of them were because of communication with MY.NET.30.3 and MY.NET.30.4. Since all traffic directed towards this IP address was alerted on, this is probably just normal communication with another university.

Link graph:

Below is a link graph of MY.NET.30.4 traffic. This host was selected because a custom Snort rule had been written to monitor the traffic of this host which signifies its importance. Only the 5 most talkative IP addresses are represented. The number in parentheses is the total number of alerts generated.



The link graph below shows four different machines communicating with one outside machine.

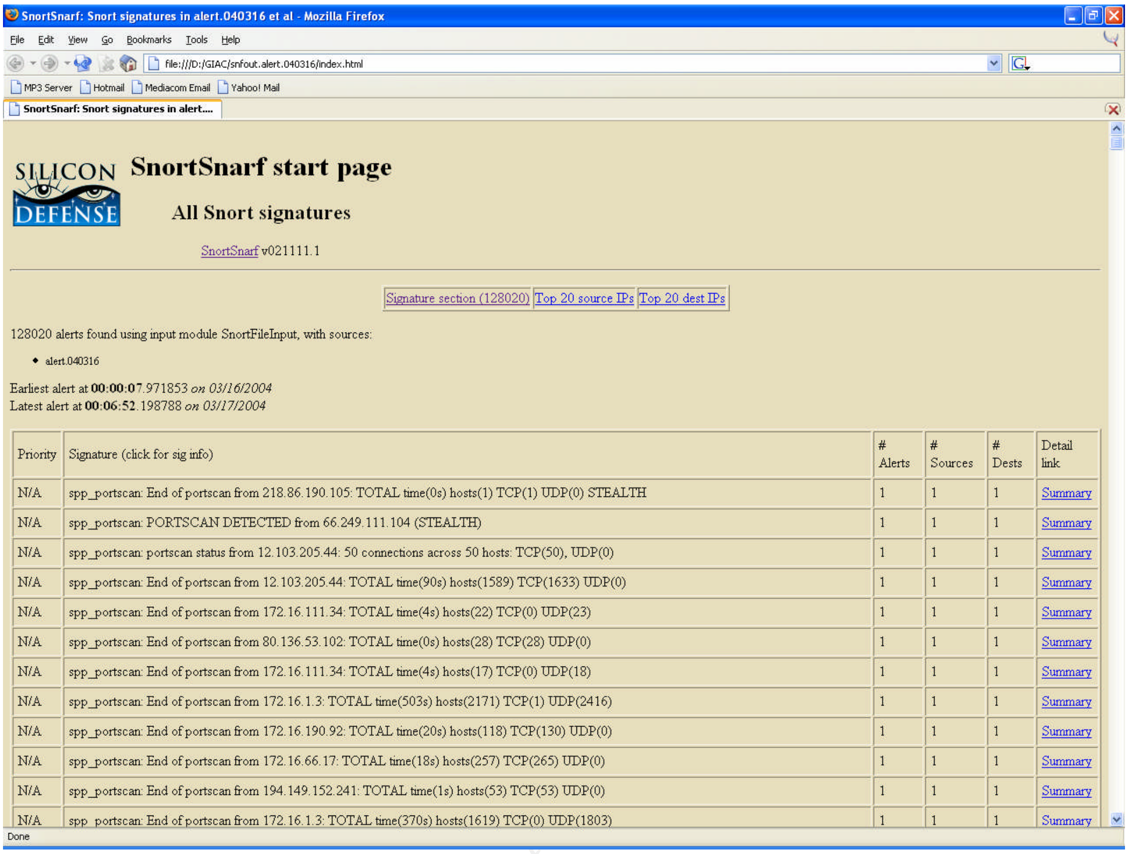


Description of analysis process

All 15 of the files were analyzed using many different applications. The files were analyzed on a UNIX system where many of the common UNIX tools were used for analysis. The files were grouped together differently to help in analyzing by attacks and by days that attacks occurred.

First the files were concatenated down to three files broken up by their type. This means that all five scan files were concatenated into one file as was the oos_report files and the alert files. Next they were concatenated down by days. This means that the oos_report file, alert file and scan file for March 16 were concatenated into one file.

To begin the initial analysis, SnortSnarf (http://www.silicondefense.com/software/snortsnarf/) was run against the alert files to produce HTML files that would aid finding security issues. SnortSnarf's output is an HTML file that can show top source IPs, top destination IPs and the alerts. Below is a screenshot of that output for one alert file.



SnortSnarf: Snort signatures in alert.040316 et al - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///D:/GIAC/snfout.alert.040316/index.html
MP3 Server
Hotmail
Mediacom Email
Yahoo! Mail
SnortSnarf: Snort signatures in alert....
SILICON DEFENSE
Done

SnortSnarf start page

All Snort signatures

SnortSnarf v021111.1

Signature section (128020) | Top 20 source IPs | Top 20 dest IPs

128020 alerts found using input module SnortFileInput, with sources:

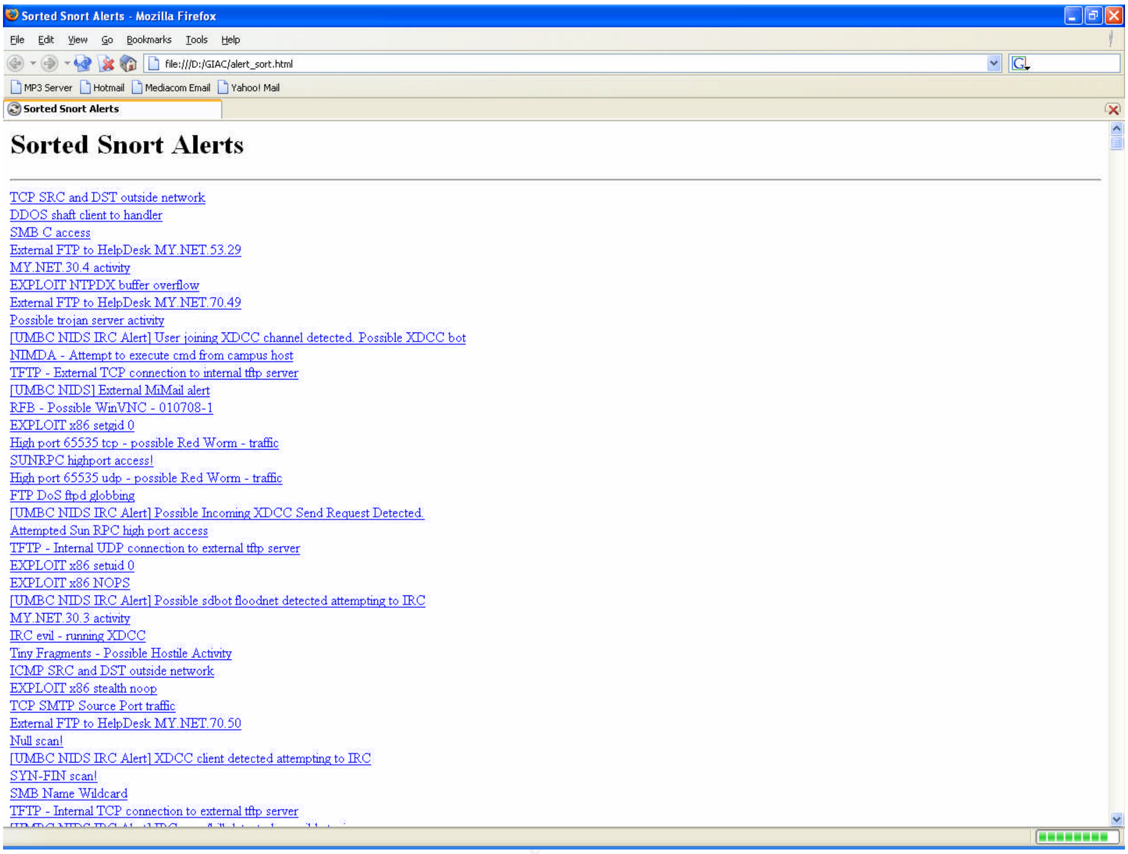
- alert.040316

Earliest alert at **00:00:07**.971853 *on 03/16/2004*
Latest alert at **00:06:52**.198788 *on 03/17/2004*

Priority	Signature (click for sig info)	# Alerts	# Sources	# Dests	Detail link
N/A	spp_portscan: End of portscan from 218.86.190.105: TOTAL time(0s) hosts(1) TCP(1) UDP(0) STEALTH	1	1	1	Summary
N/A	spp_portscan: PORTSCAN DETECTED from 66.249.111.104 (STEALTH)	1	1	1	Summary
N/A	spp_portscan: portscan status from 12.103.205.44: 50 connections across 50 hosts: TCP(50), UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 12.103.205.44: TOTAL time(90s) hosts(1589) TCP(1633) UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.111.34: TOTAL time(4s) hosts(22) TCP(0) UDP(23)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 80.136.53.102: TOTAL time(0s) hosts(28) TCP(28) UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.111.34: TOTAL time(4s) hosts(17) TCP(0) UDP(18)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.1.3: TOTAL time(503s) hosts(2171) TCP(1) UDP(2416)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.190.92: TOTAL time(20s) hosts(118) TCP(130) UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.66.17: TOTAL time(18s) hosts(257) TCP(265) UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 194.149.152.241: TOTAL time(1s) hosts(53) TCP(53) UDP(0)	1	1	1	Summary
N/A	spp_portscan: End of portscan from 172.16.1.3: TOTAL time(370s) hosts(1619) TCP(0) UDP(1803)	1	1	1	Summary

I wanted to be able to run the concatenated alert file that contained all of them, but SnortSnarf analyzes things in memory and the machine used for analysis did not have enough memory to handle an alert file of that size. For this reason SnortSnarf was run on each daily alert file only.

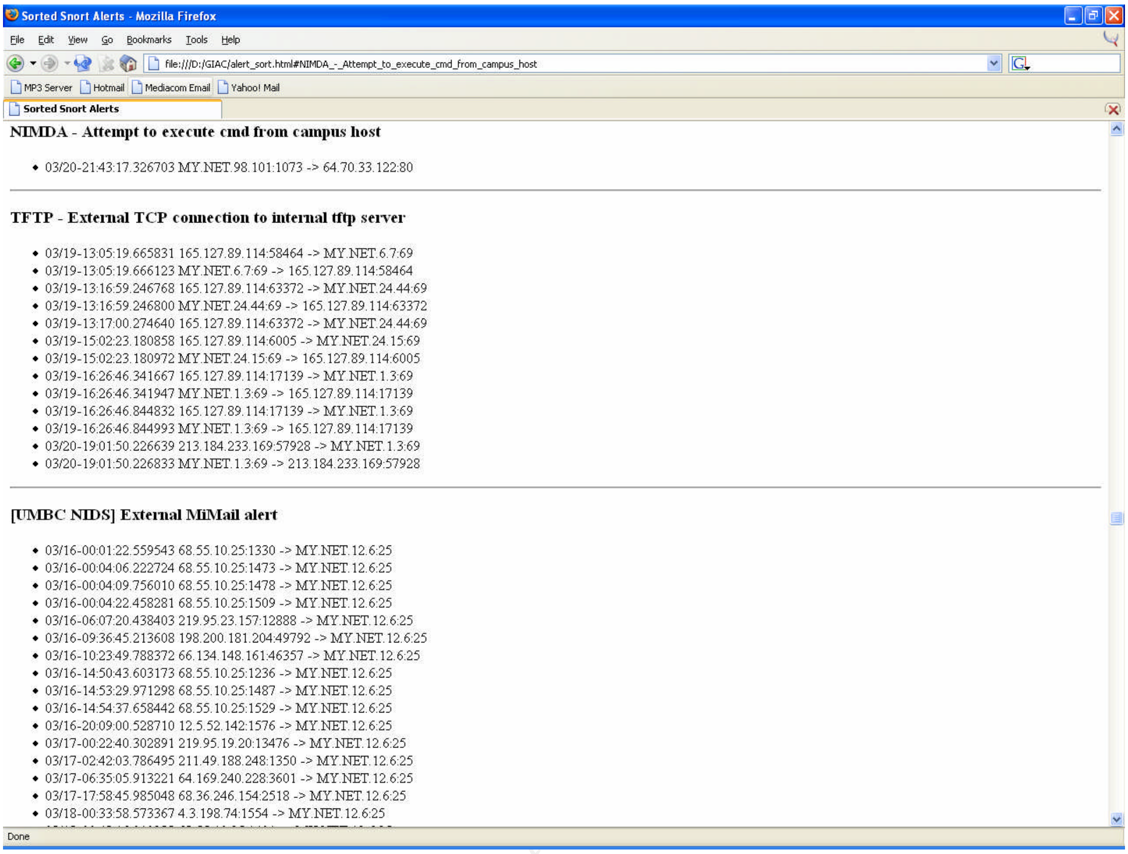
Another tool was used to analyze the complete concatenated alert file. SnortSort (http://www.dpo.uab.edu/~andrewb/snort/snort_sort.html) was used on this file and example output is shown below.



Sorted Snort Alerts

TCP SRC and DST outside network
DDOS shaft client to handler
SMB C access
External FTP to HelpDesk MY.NET.53.29
MY.NET.30.4 activity
EXPLOIT NTPDX buffer overflow
External FTP to HelpDesk MY.NET.70.49
Possible trojan server activity
[UMBC NIDS IRC Alert] User joining XDCC channel detected. Possible XDCC bot
NIMDA - Attempt to execute cmd from campus host
TFTP - External TCP connection to internal tftp server
[UMBC NIDS] External MiMail alert
RFB - Possible WinVNC - 010708-1
EXPLOIT x86 setgid 0
High port 65535 tcp - possible Red Worm - traffic
SUNRPC highport access!
High port 65535 udp - possible Red Worm - traffic
FTP DoS ftpd globbing
[UMBC NIDS IRC Alert] Possible Incoming XDCC Send Request Detected.
Attempted Sun RPC high port access
TFTP - Internal UDP connection to external tftp server
EXPLOIT x86 setuid 0
EXPLOIT x86 NOPS
[UMBC NIDS IRC Alert] Possible sdbot floodnet detected attempting to IRC
MY.NET.30.3 activity
IRC evil - running XDCC
Tiny Fragments - Possible Hostile Activity
ICMP SRC and DST outside network
EXPLOIT x86 stealth noop
TCP SMTP Source Port traffic
External FTP to HelpDesk MY.NET.70.50
Null scan!
[UMBC NIDS IRC Alert] XDCC client detected attempting to IRC
SYN-FIN scan!
SMB Name Wildcard
TFTP - Internal TCP connection to external tftp server

SnortSort groups alerts by their alert name and hyperlinks them. Clicking on an alert takes you to all the individual alerts shown here.



Sorted Snort Alerts - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
file:///D:/GIAC/alert_sort.html#NIMDA_-_Attempt_to_execute_cmd_from_campus_host
MP3 Server
Hotmail
Mediacom Email
Yahoo! Mail
Sorted Snort Alerts
Done

NIMDA - Attempt to execute cmd from campus host

- 03/20-21:43:17.326703 MY.NET.98.101:1073 -> 64.70.33.122:80

TFTP - External TCP connection to internal tftp server

- 03/19-13:05:19.665831 165.127.89.114:58464 -> MY.NET.6.7:69
- 03/19-13:05:19.666123 MY.NET.6.7:69 -> 165.127.89.114:58464
- 03/19-13:16:59.246768 165.127.89.114:63372 -> MY.NET.24.44:69
- 03/19-13:16:59.246800 MY.NET.24.44:69 -> 165.127.89.114:63372
- 03/19-13:17:00.274640 165.127.89.114:63372 -> MY.NET.24.44:69
- 03/19-15:02:23.180858 165.127.89.114:6005 -> MY.NET.24.15:69
- 03/19-15:02:23.180972 MY.NET.24.15:69 -> 165.127.89.114:6005
- 03/19-16:26:46.341667 165.127.89.114:17139 -> MY.NET.1.3:69
- 03/19-16:26:46.341947 MY.NET.1.3:69 -> 165.127.89.114:17139
- 03/19-16:26:46.844832 165.127.89.114:17139 -> MY.NET.1.3:69
- 03/19-16:26:46.844993 MY.NET.1.3:69 -> 165.127.89.114:17139
- 03/20-19:01:50.226639 213.184.233.169:57928 -> MY.NET.1.3:69
- 03/20-19:01:50.226833 MY.NET.1.3:69 -> 213.184.233.169:57928

[UMBC NIDS] External MiMail alert

- 03/16-00:01:22.559543 68.55.10.25:1330 -> MY.NET.12.6:25
- 03/16-00:04:06.222724 68.55.10.25:1473 -> MY.NET.12.6:25
- 03/16-00:04:09.756010 68.55.10.25:1478 -> MY.NET.12.6:25
- 03/16-00:04:22.458281 68.55.10.25:1509 -> MY.NET.12.6:25
- 03/16-06:07:20.438403 219.95.23.157:12888 -> MY.NET.12.6:25
- 03/16-09:36:45.213608 198.200.181.204:49792 -> MY.NET.12.6:25
- 03/16-10:23:49.788372 66.134.148.161:46357 -> MY.NET.12.6:25
- 03/16-14:50:43.603173 68.55.10.25:1236 -> MY.NET.12.6:25
- 03/16-14:53:29.971298 68.55.10.25:1487 -> MY.NET.12.6:25
- 03/16-14:54:37.658442 68.55.10.25:1529 -> MY.NET.12.6:25
- 03/16-20:09:00.528710 12.5.52.142:1576 -> MY.NET.12.6:25
- 03/17-00:22:40.302891 219.95.19.20:13476 -> MY.NET.12.6:25
- 03/17-02:42:03.786495 211.49.188.248:1350 -> MY.NET.12.6:25
- 03/17-06:35:05.913221 64.169.240.228:3601 -> MY.NET.12.6:25
- 03/17-17:58:45.985048 68.36.246.154:2518 -> MY.NET.12.6:25
- 03/18-00:33:58.573367 4.3.198.74:1554 -> MY.NET.12.6:25

This tool greatly helped in finding some of the most talkative systems and the biggest offenders as far as alerts go.
Further analysis was done using common UNIX tools. Grep, cut, uniq, awk and sort were some of the most heavily utilized applications used in the analysis. They helped to narrow down research and find very specific items. They are very efficient tools when sifting through an enormous amount of data.
All of these tools worked together to analyze events from the big picture on down to individual system traffic.

References

Bison, Rob, “Practical Assignment”, URL http://www.giac.org/practical/Rob_Bison.doc (March 2000)

Fearnow, Matt and Stearns, William, “Adore Worm” URL http://www.sans.org/y2k/adore.htm (April 12, 2001)

Fyodor (aka CyberPyschotic), “Sniff your network with Snort, a lightweight IDS”, URL http://www.daemonnews.org/199909/security.html (1999)

Gordon, Les, “What is the Q Trojan?”, URL http://www.sans.org/resources/idfaq/qtrojan.php

Heilman, Marshall, “GIAC Intrusion Detection In Depth”, URL http://www.giac.org/practical/GCIA/Marshall_Heilman_GCIA.doc.pdf (December 2003)

Intrusec, “55808 Trojan Analysis”, URL http://security-protocols.com/modules.php?name=News&file=article&sid=1528

Kuethe, Chris, “GCIA Practical Assignment, URL http://www.giac.org/practical/chris_kuethe_gcia.html#1.1

Larosa, Vjay, “LOGS: GIAC GCIA Version 3.2 Practical Detect”, URL http://cert.uni-stuttgart.de/archive/intrusions/2002/12/msg00196.htm (December 2002)

McAfee Security, “Linux/Typot”, URL http://hq.mcafeeasap.com/dispTrojan.asp?virus_k=100406

Mixter, “.mixter security”, URL http://mixter.void.ru/

Oborn, David, “SANS GCIA Practical Assignment”, URL http://www.giac.org/practical/David_Oborn_GCIA.html#detect4

Smith, Donald, “RE: tcp window size 55808 SYN packets”, URL http://cert.uni-stuttgart.de/archive/intrusions/2003/06/msg00153.html (June 2003)

Snort, “2.8 Preprocessors”, URL http://www.snort.org/docs/snort_manual/node17.html

Sophos Virus Analysis, “Linux/Adore, Linux/Red” URL http://www.sophos.com/virusinfo/analyses/linuxadore.html

Storm, Peter, “GCIA Certified Intrusion Analyst”, URL
http://www.giac.org/practical/GCIA/Pete_Storm_GCIA.pdf (November 2003)

Tridgell, Andrew, “Exploring SMB”, URL
http://us4.samba.org/samba/ftp/slides/cifs2000_tridge.pdf (May 8, 2000)

Wagner, Georg, “SubSeven Trojan 1.1”, URL
http://www.sans.org/y2k/subseven.htm

Whitehats Network Security Forums, URL
http://whitehats.com/cgi/forum/messages.cgi?bbs=get_topic&f=4&t=000005
(January 2000)

Wyman, Mike, “LOGS: GIAC GCIA Version 3.3 Practical Detect Q Backdoor”,
URL (http://cert.uni-stuttgart.de/archive/intrusions/2003/01/msg00509.html
(January 2003)

Other sites utilized for research

Geektools - WhoIs
http://www.geektools.com/whois

Google
http://www.google.com

Google Groups
http://www.google.com/grphp?hl=en&tab=wg&ie=UTF-8&oe=UTF-8&q=

IANA port numbers
http://www.iana.org/assignments/port-numbers

IEEE OUI and Company_id Assignments
http://standards.ieee.org/regauth/oui/index.shtml

Incidents.org
http://www.incidents.org